Visual Studio vs. RAD Studio
Comparativo entre os IDEs, para desenvolvedores Delphi Prism – Parte 3

Clube Delphi

Quick Update
Otimização de Commands para WPF

Boas Práticas
Simplicidade com o Padrão Builder

Ano 9 :: Edição 129 :: R$ 11,90

DevMedia

ISSN 1517990-7

dbExpress 4

Brinde

Comprando esta edição ganhe 2 vídeo-aulas:

- Delphi Prism - Login
- RAD PHP – Controle de Transportadora

Delphi Prism
Conheça os melhores recursos da linguagem – Parte 3

Biblioteca Mad
Compressão, Criptografia, Segurança, Tratamento de Strings e API

Dificuldades em encontrar conteúdo sobre .Net para iniciantes?

A DevMedia simplifica pra você!

easy .net magazine

Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

A Easy.Net não é feita em PDF ou Flash!
Ela utiliza um aplicativo 100% online.

DevMedia group

Acesse agora a edição Nº 1 totalmente grátis e comprove

www.devmedia.com.br/easynet

Easy.Net Magazine, seu curso inicial de .Net!

Sumário

Quick Update

06 – Otimização de Commands para WPF
[*Paulo Quicoli*]

Delphi

Banco de Dados

08 – dbExpress 4
A evolução do modelo de acesso a dados do Delphi
[*Fabrício Hissao Kawata*]

Novidades / Mini-Curso

16 – Delphi Prism
Melhorias da linguagem – Parte 3
[*Rodrigo Araujo*]

Expert

22 – Biblioteca madCollection
Muito além de exceptions e hooking
[*Vitor Luiz Rubio*]

Boas Práticas

Boas Práticas / Eng. de Software

32 – Padrões Criacionais
Simplicidade com Padrão Builder
[*Rafael Stavarengo*]

Easy Delphi

Easy Delphi – Mini-Curso

43 – Visual Studio vs. RAD Studio
Comparativo entre os IDEs, para desenvolvedores Delphi Prism – Parte 3
[*Rodrigo Araujo*]

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Brinde na web desta edição - 2 vídeos! Confira:

2 Vídeos

- **Curso completo Delphi Prism – Controle LOGIN do ASP.NET**
  http://www.devmedia.com.br/post-20218-Login-Desenvolvimento-Web-Curso-Completo-Delphi-Prism-Parte-86.html

- **Curso RAD PHP – Sistema de Transportadora - Parte 1**
  http://www.devmedia.com.br/post-20021-Introducao-Curso-RAD-PHP-Transportadora-Parte-1.html

Para acessar os vídeos utilize os seguintes dados
Login: **DVM.CD** *Senha:* **MXJ132**

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **DVM.CD** Senha: **MXJ132**

Clube Delphi

Ano 9 - 129ª Edição 2011 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Paulo Quicoli
pauloquicoli@devmedia.com.br

Daniel Laporte
daniel.sobrinho@devmedia.com.br

Equipe Editorial
Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

Jornalista Responsável
Kaline Dolabella - JP24185

Capa e Diagramação
Romulo Araujo - romulo@devmedia.com.br

Revisão
Caroline Velozo - carolinelopes@devmedia.com.br

Coordenação Geral
Daniella Costa - daniella@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor
A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Isabelle Macedo e Uellen Goulart – Atendimento ao Leitor
www.devmedia.com.br/mancad
(21) 3382-5038

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5038

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Cristiany Queiróz

publicidade@devmedia.com.br

EDITORIAL

O Delphi 1 em 1995 introduziu o que conhecíamos por Integrated Database Application Program Interface – IDAPI. Era basicamente um conjunto de bibliotecas DLLs que proviam uma forma unificada de acesso a banco de dados relacionais, principalmente Paradox, produto no qual era incluído. Nas versões seguintes do Delphi a IDAPI se transformou e foi incorporada no BDE – Borland Database Engine, conjunto de bibliotecas baseado em drivers para os principais bancos de dados do mercado da época, incluindo Oracle, DB2, Interbase e SQL Server. O BDE podia ser usado tanto por aplicações Delphi Win32 quanto C++ Builder, ferramenta também da Borland análoga ao Delphi, com o mesmo IDE e framework (VCL), porém com a sintaxe da linguagem baseada no C. O BDE tinha inúmeros problemas bastante conhecidos desde os seus primórdios, que não atenderam a demanda de requisitos básicos que surgiram com a crescente evolução dos sistemas corporativos, baseados em multicamadas (n-tier) e Web. Não era escalável, possui vários bugs com acesso concorrentes, seu mecanismo de update cache era arcaico (em disco, não em memória, pois na época sobrava HD e faltava RAM), existiam bugs conhecidos com alguns tipos de dados de alguns bancos (BLOBs, BCD), tinha problemas quando usado com o MTS/COM+, rodava somente em ambiente Windows (o que era um problema com a política da atual Borland na época de construir ferramentas multiplataforma), além disso era necessário instalar praticamente todo o engine nos clientes mesmo que fosse acessar um simples arquivo DBF. Até era possível levar apenas um driver, mas o processo era complicado. O BDE não rodava bem em rede, o que começou a ser um problema já no Delphi 2, quando surgiram os primeiros conceitos de desenvolvimento client-server. O BDE prendia o cursor de dados do banco, consumindo recursos. No Delphi 5 a Borland lança em seu IDE dois novos frameworks alternativos de acesso a dados, o DbGo, componentes que encapsulam as tecnologias ADO, ODBC e OleDB, e o IBX, para acesso exclusivo e nativo ao Interbase (e algumas versões do Firebird). Ambos também tinham seus defeitos, mas pelo menos tínhamos alternativas (outra era a Zeos, não nativa). O Kylix 1, o Delphi para Linux, trouxe à tona o dbExpress, com a mesma ideia do BDE, um conjunto de componentes bastante semelhantes (baseados em TDataSet), porém com uma arquitetura interna diferente. Usando interfaces e drivers, através de uma camada extremamente fina, era possível fazer uma aplicação se comunicar diretamente com o banco alvo. A cache passou a ficar em memória, no ClientDataSet, sem prender cursores do banco. O framework era multiplaforma, tanto que apareceu no ambiente Windows com o Delphi 6, e está hoje entre nós até o Delphi XE. Ele até deu uns passeios pelo mundo .NET com uma implementação para ADO.NET chamada BDP, mas que se foi junto com o Delphi for .NET (hoje usamos os Prism e os providers nativos do .NET Framework). E desde o Delphi 2009, o DBX faz também parte do lado cliente das aplicações DataSnap, unificando também os diferentes protocolos de acesso. Sem dúvida uma enorme evolução, que você vai poder acompanhar nesta edição, no artigo do Fabrício Kawata, que demonstra os principais recursos do novo dbExpress 4.

Guinther Pauli
guinther@devmedia.com.br
Certificado Delphi 3,5,6,7,2005,2006,Web,Linux
Microsoft Certified: MCP, MCAD, MCSD.NET, MCTS, MCPD (ASP.NET,C#)
guintherpauli.blogspot.com - twitter.com/guintherpauli

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

ClubeDelphi PLUS

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

DevMedia

Sistema de créditos DevMedia

Conteúdo **completo** ao seu alcance

Com o sistema de créditos DevMedia você pode comprar individualmente mais de 4.000 vídeo aulas, 8000 artigos, além de palestras e vídeos com formações completas.

TODAS as vídeo aulas que você sempre quis ver a partir de

R$ 0,99

ECONOMIA: você não precisa assinar 8 produtos diferentes para ter acesso ao conteúdo completo do site!

Saiba mais sobre o Sistema de Créditos!

www.devmedia.com.br/creditos

DevMedia

Otimização de Comm

O uso do WPF vem crescendo grandemente aqui no Brasil para desenvolvimento de aplicativos desktop para Windows. Com isso, desenvolvedores têm corrido atrás de aprendizado rápido e consequentemente na grande maioria das vezes não conseguem obter conhecimento sobre boas práticas relacionadas à tecnologia. Com isso acabam utilizando recursos de maneira não otimizada.

A razão de um comando é separar o que fazer, de quem vai fazer. Por exemplo um comando Copiar de um editor de texto qualquer. A lógica de como esse copiar é feito fica encapsulada em uma classe que implementa a interface ICommand. E esse comando pode ser associado a controles visuais que irão executá-lo. Pensando no editor de texto, o mesmo comando pode ser executado por um menu Editar>Copiar, por um atalho CTRL+C ou por um botão em alguma barra de ferramentas. Assim existe a reutilização de código. A ideia é similar aos Actions do Delphi.

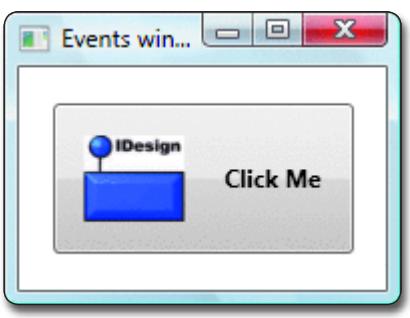


Figura 1. Aplicativo simples

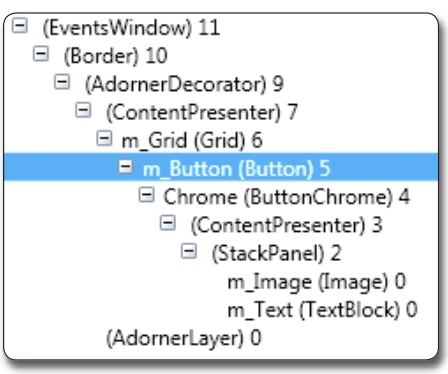


Figura 2. Visual Tree

Routed Commands

O WPF implementa esse conceito de comando através dos chamados Routed Commands, ou comandos roteáveis. Como é sabido, a estrutura visual do WPF/Silverlight é separada em Visual Tree e Logical Tree. Para entender vamos observar a Visual Tree da janela exibida na **Figura 1**.

Essa simples janela, com um botão que possui uma imagem interna criou a seguinte Visual Tree da **Figura 2**.

Essa é a árvore dos elementos que compõem a visualização, mesmo que não sejam vistos. Como é possível notar, existe janela e nela uma borda. Ela é composta por um AdornerDecorator e dentro dele há um ContentPresenter. Dentro deste existe um Grid e dentro dele um Button. E por aí prossegue a hierarquia dos elementos. Mas o que tem tudo isso a ver com Routed Commands?

Command Manager

Quando você cria um routed command, ele é associado a um agente da estrutura do WPF que fica em background verificando se o determinado comando pode ser executado ou não. Essa verificação é feita percorrendo toda visual tree até que se encontre o elemento que está associado ao comando. Então imaginando que essa verificação se dá a cada meio segundo, a cada meio segundo a visual tree será varrida. Como pode ser visto na **Figura 2**, para uma janela simples já temos uma certa quantidade de elementos, pare para pensar agora naquela sua tela de cadastro enorme. Qual seria a quantidade de elementos existentes e qual o impacto de performance em percorrer essa árvore a cada meio segundo?

Boas práticas

Pensando nisso a comunidade de usuários e a própria equipe de desenvolvimento, na época, arquiteto do WPF, Josh Smith pensou em uma classe que também implementaria a interface ICommand – já que essa é uma exigência para comandos – mas que não faria com que a visual tree fosse varrida e que também fosse mais simples de se implementar pelo desenvolvedor. Na verdade esse novo comando iria ligar a interface a um delegate, de forma direta. Depois que essa ideia foi publicada, vários desenvolvedores fizeram suas implementações criando os chamados RelayCommands, DelegatingCommands e outros que são fáceis encontrar nos mais variados frameworks MVVM. Inclusive o padrão Model-View-ViewModel também foi criado por Josh e abraçado pela comunidade. Para demonstração desse formato de comando utilizou-se o framework MVVM Helpers.

A definição de um comando

O exemplo é bem simples. Temos uma janela que possui um TextBox e um botão. Esse botão está ligado a um comando que só será executado caso o TextBox esteja preenchido com mais de dois caracteres. Para separar a lógica da apresentação, foi criado um ViewModel representando o texto que pode ser digitado e o comando que é associado ao botão, como é mostrado no exemplo em Delphi Prism da **Listagem 1**.

O comando é só uma propriedade do tipo DelegatingCommand do MvvmHelpers que implementa ICommand, é preciso associar a ele uma implementação do que

ands para WPF

Paulo Quicoli

fazer e uma verificação se ele pode executar sua função. Isso é feito no construtor do ViewModel (**Listagem 2**).

Como pode ser visto a ideia é simples. Um comando é definido e a ele são associados dois métodos, um indicando o que fazer e outro que permite ou não essa execução. Automaticamente o comando é registrado no Command Manager do WPF, porém agora quando for verificado se o comando pode ser executado, a visual tree não será percorrida, mas sim o método CanExibirMensagem que será chamado de forma direta. A **Listagem 3** mostra como esse comando foi atribuído ao botão. Mais uma vez o WPF/Silverlight mostra sua flexibilidade e facilidade. Através de seu mecanismo de data binding, de ligação de dados, foi fácil dizer ao botão para executar o comando criado.

Ao executar o aplicativo é possível ver o botão ser habilitado ou desabilitado de forma automática.

Listagem 1. JanelaViewModel

```
type
 JanelaViewModel = public class
 public
    property Texto: String; notify;
    property CmdExibirMensagem:
    DelegatingCommand;
    constructor;
 end;
```

Listagem 2. Implementando o comando

```
constructor JanelaViewModel;
var
   ExibirMensagem: Action<Object>;
   CanExibirMensagem: Func<Object,boolean>;
begin
 ExibirMensagem := s->
 begin
    MessageBox.Show('Teste');
 end;

 CanExibirMensagem := y->
 begin
  exit Texto.Length >2;
 end;

 CmdExibirMensagem := new DelegatingCommand
 (ExibirMensagem, CanExibirMensagem, false);

 Texto := string.Empty;
end;
```

Listagem 3. Data binding em ação

```
    <TextBox
      Text="{Binding Texto,Mode=TwoWay,Update
      SourceTrigger=PropertyChanged}"
      Height="24"
      Width="150"/>
    <Button
      Content="Clique aqui"
      Width="100" Height="24"
      Command="{Binding CmdExibir
      Mensagem}"/>
```

Listagem 4. Comando fora do Command Manager

```
CmdExibirMensagem := new DelegatingCommand
(ExibirMensagem, CanExibirMensagem, false);

property Texto: String read _texto write SetTexto;
notify;

procedure JanelaViewModel.SetTexto(value:
String);
begin
   _texto := value;
   CmdExibirMensagem.RaiseCanExecuteChanged;
end;
```

Otimização[2]

Ao fazer uso desse tipo de comando ganhamos desempenho. Sem dúvida. Porém o Command Manager também gera um processamento às vezes desnecessário. Para provar isso experimente colocar um breakpoint no método CanExibirMensagem e rodar o aplicativo. Não é possível efetuar um debug da execução porque a todo o momento o Command Manager fica verificando a execução do comando.

Mais uma vez a comunidade pensou numa solução para isso e desenvolveu uma saída. Existe a possibilidade de não registrar o comando no Command Manager. Assim essa verificação não é feita de forma automática e constante, cabendo a nós chamar a verificação quando for necessário.

Neste caso, a regra para saber se o comando pode ser executado está na verificação da propriedade Texto. Assim, sempre que essa propriedade for alterada, vamos disparar a verificação. Na **Listagem 4** temos a alteração na implementação do comando. Foi necessário apenas passar o parâmetro False para que ele não seja registrado Command Manager. E na sequência, no método set da propriedade Texto é chamada a verificação.

Para testar o funcionamento basta colocar novamente um breakpoint no método CanExibirMensagem. Ao executar a aplicação a parada do breakpoint só irá acontecer quando a propriedade Texto sofrer alguma alteração.

Paulo Quicoli
pauloquicoli@gmail.com
Editor técnico das revistas ClubeDelphi, .NET Magazine e WebMobile Magazine. Formado em processamento de dados pela FATEC-TQ. Atua como Arquiteto de sistemas .NET da ControlM Informática (www.controlm.com.br). Utiliza Delphi desde sua primeira versão, entusiasta do desenvolvimento orientado a objetos, tem publicado vários artigos sobre o assunto. Blog: http://quicoli.wordpress.com

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

dbExpress 4

A evolução do modelo de acesso a

Historicamente, a arquitetura dbExpress surgiu no Delphi ainda na versão 6 do IDE, como uma alternativa ao até então predominante BDE (Borland Database Engine), responsável pela conectividade entre aplicações e banco de dados. Desde seu lançamento, a aceitação do dbExpress foi tamanha que na versão posterior (Delphi 7) ele já se consolidava como uma das tecnologias mais usadas pela comunidade de desenvolvedores da linguagem em aplicações envolvendo banco de dados.

Posteriormente, ainda no Delphi 2006, o dbExpress se encontrava na versão 3. Porém foi com o lançamento do Delphi 2007 que sua arquitetura sofreu uma grande mudança, tendo seu núcleo (core) totalmente remodelado. Suas bibliotecas e drivers foram reescritos para 100% código Delphi, diferente da escrita anterior em C/C++. Dessa forma, novos drivers também puderam ser escritos na linguagem Delphi. Todas estas mudanças culminaram numa nova versão do dbExpress, de número 4, que representou um marco na história do framework.

Já em seu release seguinte, no Delphi 2009, com o framework já estável, poucas foram as melhorias apresentadas. Destaque para o lançamento do novo driver cliente para DataSnap. Este recurso aliás, resultou numa grande revolução em termos de desenvolvimento multicamadas no Delphi, conforme você pôde conferir aqui mesmo nesta revista em artigos específicos. No Delphi 2010 a arquitetura do dbExpress passou por pequenas alterações que, basicamente, afetaram sua parte de drivers. Em síntese, foi atualizada a sua compatibilidade às novas versões dos bancos de dados suportados até então (como por exemplo: Interbase, Oracle, MS SQL Server, MySQL), bem como a forma de acesso a eles, além do suporte a novos SGBDs cujo destaque foi o tão aguardado suporte ao Firebird. Funcionalidade esta, aliás, que era um pedido antigo da comunidade em razão do casamento (Firebird e dbExpress) empregado em grande parte das aplicações VCL Win32 desenvolvidas. Apenas para mencionar, apesar do "casamento perfeito" já percebido desde versões muito anteriores do IDE, o Delphi demorou a oferecer suporte completo ao Firebird em vista de desentendimentos ocasionais entre a então Borland e a Firebird Foundation.

Resumo DevMan

De que trata o artigo:

Neste artigo você verá um apanhado geral de toda a evolução de um dos mais bem sucedidos framework que acompanha o Delphi desde sua versão 6, o dbExpress. Ele, através de seu conjunto de classes estabelece a comunicação das aplicações com diversos bancos de dados e até mesmo Servidores de Aplicação, através do uso de drivers exclusivos. Em vista disso, o leitor será situado de onde se encontra o framework atualmente, num demonstrativo de toda trajetória de mudanças e novidades até sua versão atual, de número 4.

Para que serve:

A utilização do framework DBX no Delphi se dá por meio do acesso direto às suas classes ou ainda através de seus componentes da VCL, presentes no grupo dbExpress da Tool Palette da IDE. Ao acompanhar todo um histórico do framework que você utiliza, faz com que seus conceitos atuais sejam reforçados, além da possibilidade de novos serem adicionados ao seu conhecimento.

Em que situação o tema é útil:

O framework dbExpress, através de seu conjunto de classes abstratas e drivers, provê uma comunicação leve da aplicação com um servidor, seja este de banco de dados ou de aplicação (mais recentemente). Alguns exemplos de bancos nativamente suportados são: Interbase, Firebird, Oracle e SQL Server. Dito isto, logo se vê a importância do framework e o motivo pelo qual o mesmo é tão utilizado nas diversas aplicações Delphi espalhadas pelo globo. Dessa forma, todo estudo dos bastidores do framework se faz de importância, principalmente para ganho de conhecimento, como é o caso deste artigo.

dbExpress 4:

Neste artigo será apresentado o framework dbExpress, também conhecido na comunidade Delphi como simplesmente DBX. Isso se dará de forma conceitual, abrangendo seu histórico, funcionalidade e toda sua evolução que pôde ser percebida no decorrer dos lançamentos de cada nova versão do Delphi. Exposto isso, ficará claro o motivo pelo qual este framework se tornou tão popular e um dos mais utilizados em soluções Delphi que envolvem conectividade. Indo além, pode ser acrescido à sua robustez o fato dele ser algo nativo do IDE, fazendo suas aplicações ficarem independentes, por exemplo, de drivers de terceiros para conectividade com banco de dados.

dados do Delphi

Fabrício Hissao Kawata

Mas em síntese, desde que o framework foi batizado como dbExpress 4 (ou DBX 4), se manteve estável e sofreu apenas mudanças pontuais, não havendo mais grandes alterações em sua arquitetura até os dias atuais na versão mais recente do IDE Delphi, o XE.

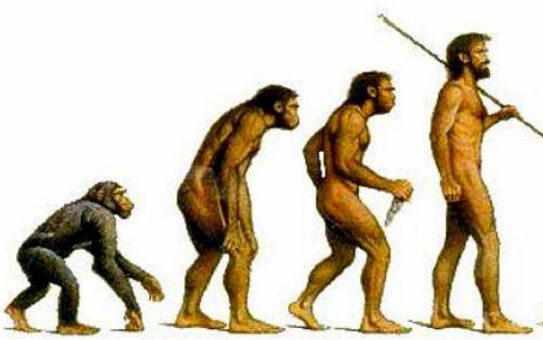
Reforçando, o presente artigo se apoiará em vários pontos na linha do tempo de evolução do dbExpress, com o objetivo de verificar como o framework alcançou um alto grau de maturidade ao longo da evolução da própria IDE, em vista do surgimento gradativo das necessidades da comunidade de desenvolvedores Delphi. Obviamente este trabalho não tem a intenção de mostrar exatamente todas as mudanças que surgiram desde sua primeira versão, mas sim as de maiores destaque.

A unit DBXCommon

Com a total reescrita de sua arquitetura para a linguagem Delphi, o dbExpress 4 trouxe consigo uma nova unit que concentra as principais classes do framework, responsáveis, por exemplo, pela parte de transações, conexões, propriedades, drivers, entre outros recursos. Consequentemente, através delas, tornaram-se mais simples a escrita de novos drivers para a arquitetura.

A integração da unit DBXCommon com as aplicações se dá por meio direto, através do uso de suas classes, ou ainda pelos componentes da VCL do grupo dbExpress presentes na Tool Palette do Delphi. Vale lembrar que a VCL (Visual Component Library) é uma estrutura orientada a objetos baseada em componentes visuais para a construção de aplicativos da plataforma Windows, da Microsoft. Foi desenvolvida pela Borland, sendo toda escrita em Object Pascal. Dentre suas funcionalidades, a principal é sua capacidade de proporcionar a comunicação de uma aplicação cliente com um servidor de banco de dados ou, mais recentemente, com um servidor de aplicação DataSnap, utilizando para isso drivers nativos. O DataSnap, em seu estágio atual pode ser definido como a tecnologia Delphi para a criação de aplicações multicamadas (N-tier) numa estrutura Cliente-Servidor. Seu principal atrativo é proporcionar de forma transparente a capacidade da parte cliente invocar métodos remotos definidos no Servidor. A comunicação entre as partes se dá por meio de rede local ou internet. Sua lista de drivers é extensa, com suporte para: Interbase, Firebird, DB2, Oracle, MySQL, SQL Server, DataSnap e outros.

Logo, os componentes dbExpress VCL possuem na implementação de sua estrutura interna alguns objetos descendentes classes presentes em DBXCommon. Podemos até assim dizer que esses objetos são de mais "baixo nível", uma vez que os componentes em si apenas os encapsulam. Como exemplo, tomemos as classes TDBXConnection e TDBXCommand. A primeira representa uma conexão dbExpress válida, direcionada para determinado driver do framework. Através dela são nos fornecidas algumas ações sobre o banco de dados, como criar comandos, recuperar registros e metadados, além de gerenciamento de transações. Já a segunda é a classe que irá executar efetivamente as instruções SQL e Stored Procedures que lhe forem passadas.



Nota do DevMan

BDE é a sigla para Borland Database Engine. Pode ser definida como uma arquitetura Windows para conectividade entre aplicações nativas (Ex: Delphi, C++ Builder) e banco de dados (Ex: Paradox, dBASE). Na prática, uma aplicação pode possuir vários objetos BDE, sendo os responsáveis pela manipulação das entidades do banco de dados, como por exemplo, suas tabelas.

SGBD, acrônimo de Sistema de Gerenciamento de Banco de Dados, é um agrupamento de programas que visam proporcionar o armazenamento e manipulação das informações de um banco de dados. Exemplos de SGBDs: Interbase, Firebird, Oracle, MySQL, Microsoft SQL Server, etc.

O Firebird é um Sistema Gerenciador de Banco de Dados open source, originado a partir do SGBD Interbase, da então Borland, em meados de 2000. Foi elaborado por uma comunidade de desenvolvedores que o mantém até os dias de hoje. Um de seus grandes atrativos, além de ser gratuito, é que ele atende múltiplas plataformas, sendo capaz de rodar em sistemas operacionais Windows, Linux e Unix.

Nota do DevMan

ADO.NET pode ser definido como um conjunto de classes ou assemblies, presentes no Microsoft .Net Framework, cujo principal objetivo é prover comunicação com banco de dados. Mais especificamente aos seus dados armazenados, realizando operações de leitura e atualização. Exemplos de classes ADO.NET: SqlConnection, SqlCommand, SqlDataReader e SqlParameter.

Nota do DevMan

Delphi Prism é a ferramenta da Embarcadero para desenvolvimento de aplicações para .NET Framework da Microsoft e Mono da RemObjects. Utiliza como linguagem o Oxygene, que possui um alto grau de semelhança com a linguagem utilizada no Delphi. O Delphi Prism se integra ao IDE do Microsoft Visual Studio em forma de plug-in, com o intuito de aproveitar ao máximo os recursos do .NET Framework.

Exemplo prático

Para facilitar o entendimento, na **Listagem 1** temos um exemplo real da implementação destas classes de DBXCommon, onde é executado um comando de Update no banco de dados definido na conexão dbExpress em questão. Esta por sua vez é obtida através da propriedade DBXConnection do componente TSQLConnection (pressupondo que o projeto já possua um), conforme pode ser visto na linha 10.

Neste trecho de código ao invés de se utilizar componentes VCL, optou-se por fazer uso das classes do framework dbExpress para se executar um comando de atualização na tabela EMPLOYEE no banco Employees do Firebird. Como mencionado, a classe TDBXConnection é quem provê a criação de comandos, portanto, na linha 13 pedimos a ela que nos retorne um objeto TDBXCommand. Com o objeto criado, é necessário apenas que se passe o comando SQL à sua propriedade Text para que este seja executado no banco de dados da conexão. É imprescindível ressaltar que para estes casos onde são utilizadas as classes dbExpress manualmente, deve-se destruir o objeto Command sempre depois de usá-lo.

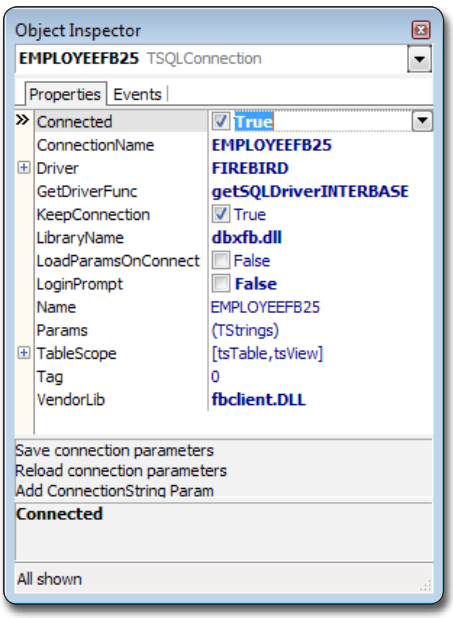


Figura 1. Propriedades de uma conexão dbExpress Firebird

Listagem 1. Executando um comando

```
01  procedure TfrmDBXCommon.btnExecuteCommandClick(Sender: TObject);
02  var
03    Con: TDBXConnection;
04    Cmd: TDBXCommand;
05  const
06    SQL: string = 'update EMPLOYEE set LAST_NAME = ''TESTE'' +
07                  'where EMP_NO = 2';
08  begin
09    SQLConnection1.Connected := True;
10    Con := SQLConnection1.DBXConnection;
11
12    try
13      Cmd := Con.CreateCommand;
14      Cmd.Text := SQL;
15      Cmd.ExecuteUpdate;
16      ShowMessage('Comando executado com sucesso!!!');
17    finally
18      FreeAndNil(Cmd);
19    end;
20  end;
```

Uma curiosidade: para quem está habituado com a tecnologia .Net, pode ter percebido certa semelhança entre as classes mencionadas com as da arquitetura ADO.NET. Isto é plausível, uma vez que com a reestruturação do dbExpress 4 a partir do Delphi 2007, a CodeGear (então fabricante do Delphi) visou a integração entre estas duas tecnologias. A idéia era permitir que classes escritas com dbExpress no Delphi Win32 pudessem também serem compiladas e re-utilizadas em projetos Delphi.Net. Lembrando que na época, o IDE ainda oferecia suporte para projetos Delphi Win32 e Delphi.Net, sendo este último descontinuado e posteriormente substituído pelo Delphi Prism.

Driver Firebird

Em termos práticos, a dificuldade encontrada até o Delphi 2009 pelos desenvolvedores Delphi era que ao se utilizar o dbExpress como tecnologia de acesso a um banco de dados Firebird, não se tinha um driver adequado para ser utilizado pela conexão. Dessa forma fez-se necessária a busca por soluções. Dentre estas, a solução encontrada na maioria dos casos era a utilização de um driver de terceiro, geralmente pago, ou ainda fazendo uso do driver para Interbase, que não era totalmente compatível com o Firebird causando problemas inesperados (por exemplo, em campos do tipo Blob).

Mesmo sabendo da necessidade da comunidade, a então fabricante do Delphi na época por questões já mencionadas anteriormente neste artigo, não disponibilizava um driver adequado. Todavia, com o IDE passando para as mãos da Embarcadero, a tensão que havia entre as fabricantes diminuíra. Iniciava-se assim pela primeira vez o planejamento para o suporte completo ao Firebird, que culminou no lançamento oficial, no Delphi 2010, de um driver dbExpress específico. Este novo driver oferece suporte ao Firebird desde sua versão 1.5. Na **Figura 1** pode ser visto um componente TSQLConnection configurado para uma conexão com o banco de exemplo Employee do Firebird.

Neste exemplo, na propriedade DriverName do componente, vemos a indicação do novo driver suportado pelo dbExpress. Em LibrayName, temos a menção à dll do novo driver Firebird. Por fim, em VendorLib temos a indicação da biblioteca cliente exclusiva do SGBD, fbclient.dll. Ao se fazer uso deste novo driver automaticamente é adicionada uma referência à unit DBXFirebird, específica para o gerenciamento dos parâmetros deste tipo de conexão.

Em função do completo suporte ao Firebird, o dbExpress proporciona inclusive a manipulação dos metadados deste SGBD, tal como a criação tabelas, índices, chaves etc. Isso se dá através das units DBXDataExpressMetaDataProvider e DBXMetaDataProvider. A seguir, na **Listagem 2**, você confere um exemplo de seu uso para a criação de uma nova tabela no banco de dados Employee. A tabela em questão irá se chamar Usuários e terá três campos: Código, do tipo Integer e que será sua chave primária, Nome, do tipo Varchar de tamanho 50 e Admin, do tipo booleano.

No código desta listagem pode-se perceber entre as linhas 1 e 8 que para cada tipo de variável é relacionado um tipo oferecido pelo dbExpress. Por exemplo, a variável Tabela é do tipo TDBXMetaDataTable, que é a classe usada pelo framework para se criar uma tabela no banco de dados, via código. A variável FieldInteger que representa o campo do tipo inteiro da tabela, será do

Listagem 2. Criando uma tabela

```
01  var
02    Tabela: TDBXMetaDataTable;
03    PrimaryKey: TDBXMetaDataIndex;
04    FieldInteger: TDBXInt32Column;
05    FieldString: TDBXAnsiVarCharColumn;
06    FieldBool: TDBXBooleanColumn;
07    Provider: TDBXDataExpressMetaDataProvider;
08  begin
09    //dar uses em DBXDataExpressMetaDataProvider e DBXMetaDataProvider
10    try
11      Provider := TDBXDataExpressMetaDataProvider.Create;
12      Provider.Connection := EMPLOYEEFB25.DBXConnection;
13      Provider.Open;
14
15      Tabela := TDBXMetaDataTable.Create;
16      Tabela.TableName := 'USUARIOS';
17
18      FieldInteger := TDBXInt32Column.Create('CODIGO');
19      FieldInteger.Nullable := False;
20
21      FieldString := TDBXAnsiVarCharColumn.Create('NOME', 50);
22      FieldString.Nullable := False;
23
24      FieldBool := TDBXBooleanColumn.Create('ADMIN');
25      FieldString.Nullable := False;
26
27      Tabela.AddColumn(FieldInteger);
28      Tabela.AddColumn(FieldString);
29      Tabela.AddColumn(FieldBool);
30
31      Provider.CreateTable(Tabela);
32
33      PrimaryKey := TDBXMetaDataIndex.Create;
34      PrimaryKey.TableName := 'USUARIOS';
35      PrimaryKey.AddColumn(FieldInteger.ColumnName);
36      Provider.CreatePrimaryKey(PrimaryKey);
37
38      ShowMessage('Tabela criada com sucesso!!!');
39    finally
40      FreeAndNil(PrimaryKey);
41      FreeAndNil(Tabela);
42      FreeAndNil(Provider);
43    end;
44  end;
```

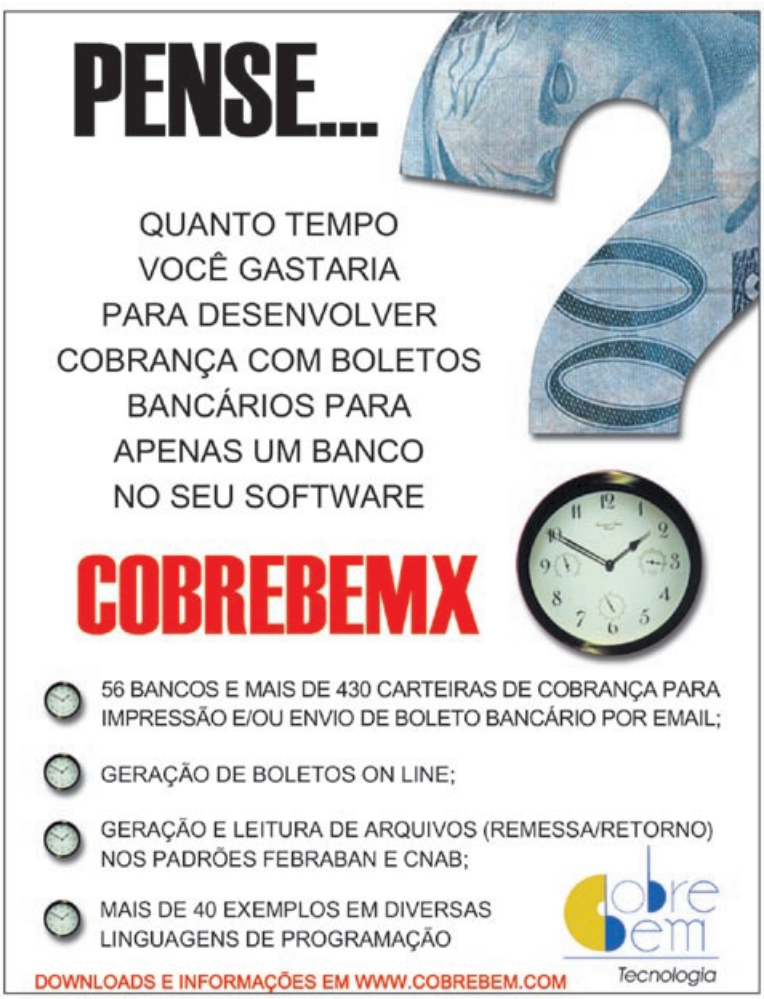
tipo TDBXInt32Column e assim sucessivamente para os outros. Além disso, existe ainda a variável Provider, do tipo TDBXDataExpressMetaDataProvider que nesta estrutura irá prover uma conexão Firebird válida para o banco onde a tabela será efetivamente criada. Todos os campos tiveram sua propriedade Nullable assinalada como False para não permitir valores nulos em suas colunas, ou seja, estamos definindo os campos como NOT NULL. Por fim, para se criar a tabela fisicamente no banco de dados, utiliza-se o método CreateTable de um objeto Provider.

Driver SQL Server

Como mencionado anteriormente, no dbExpress 4 alguns de seus drivers foram atualizados para atender as versões mais recentes da sua lista de bancos de dados suportados, por exemplo: Oracle 11g, Interbase 2009/XE, MySQL 5.x, entre outros. Dentre os drivers existentes, o único a sofrer uma considerável mudança em sua arquitetura foi o driver para SQL Server. A explicação para isso é que até então o driver utilizava um data provider OLEDB para a comunicação com o SGBD. Porém, este provider só suportava recursos até a versão 2000 do SQL Server. A partir do dbExpress 4 o driver então passou a utilizar o cliente nativo Microsoft SQL Server 2008 Native Client para suportar as versões mais atuais do SGBD (2005, 2008 e futuras).

Controle transacional

Anteriormente ao DBX 4 o controle de transações dbExpress numa aplicação era feito conforme a **Listagem 3**,



Nota do DevMan

DLL é a abreviação de Dynamic Link Library. Em português pode ser traduzido como biblioteca de ligação dinâmica e pode ser comparado a um arquivo executável, já que pode conter códigos, dados e recursos em sua estrutura. Tais recursos podem ser compartilhados com outros programas, permitindo a chamada de suas funcionalidades internas. Além disso, também é comumente utilizada para a modularização de aplicações, onde cada DLL abrange uma responsabilidade do sistema.

Listagem 3. Controle transacional dbExpress até a versão 3

```
01 var
02  T:TTransactionDesc;
03 const
04  SQL: string = 'update EMPLOYEE set LAST_
    NAME = ''TESTE2''' +
05   'where EMP_NO = 2';
06 begin
07  SQLConnection1.Connected := True;
08  T.TransactionID := 1;
09  T.IsolationLevel := xilREADCOMMITTED;
10  SQLConnection1.StartTransaction(T);
11  try
12   SQLConnection1.ExecuteDirect(SQL);
13   SQLConnection1.Commit(T);
14  except
15   SQLConnection1.Rollback(T);
16 end;
17 end;
```

Listagem 4. Controle transacional dbExpress na versão 4

```
01 var
02  Trans:TDBXTransaction; //dar uses em
    DBXCommon
03 const
04  SQL: string = 'update EMPLOYEE set LAST_
    NAME = ''TESTE3''' +
05          'where EMP_NO = 2';
06 begin
07  SQLConnection1.Connected := True;
08  try
09   Trans := SQLConnection1.BeginTransaction(
10    TDBXIsolations.ReadCommitted);
11   SQLConnection1.ExecuteDirect(SQL);
12   SQLConnection1.CommitFreeAndNil(Trans);
13  except
14   SQLConnection1.RollbackFreeAndNil(Trans);
15  end;
16 end;
```

onde logo de início era necessário um objeto do tipo TTransactionDesc, que é um record que descreve uma transação em si. Nesta estrutura existem dois itens primordiais que deveriam ser definidos no escopo do código: TransactionID e IsolationLevel. Com base no código, na linha 8 temos a atribuição de TransactionID que, conforme o nome sugere é um número identificador da transação, geralmente definido aleatoriamente. Na linha 9 é definida a IsolationLevel, que é o nível de isolamento da transação, podendo ser ela ReadCommitted, DirtyRead, RepeatableRead ou Custom. Uma vez definida a estrutura do objeto TTransactionDesc, passa-se um ponteiro deste para o método StartTransaction da conexão, que irá efetivamente iniciar a transação. Por fim, para se confirmar ou reverter uma transação iniciada eram utilizados os métodos Commit e Rollback, respectivamente.

Com o dbExpress 4 essa forma de lidar com transações foi remodelada com o acréscimo de alguns novos métodos, como também a não necessidade de se definir manualmente um identificador (ID) para cada transação criada. Para ficar mais fácil a visualização das mudanças, tomemos o mesmo código que foi reescrito para a versão 4 do dbExpress (**Listagem 4**).

Nesta **Listagem 4** percebe-se logo de início a ausência de um objeto do tipo TTransactionDesc, que não se faz mais necessário, uma vez que toda a parte transacional será provida pela nova classe TDBXTransaction que está contida na unit DBXCommon. Portanto, veja que na linha 9 é instanciado o objeto de transação utilizando para isso a função BeginTransaction da conexão. BeginTransaction é um novo método do componente TSQLConnection que inicia uma transação no banco de dados da conexão, cujo retorno é um objeto do tipo TDBXTransaction. Ainda neste mesmo método é passado como parâmetro o nível de isolamento desejado para a transação que está sendo iniciada. Para terminar, os métodos Commit e Rollback do dbExpress 3 foram substituídos pelos novos CommitFreeAndNil e RollbackFreeAndNil, que têm a mesma incumbência de seus antecessores, porém com o aditivo de se liberar automaticamente da memória a transação utilizada no momento.

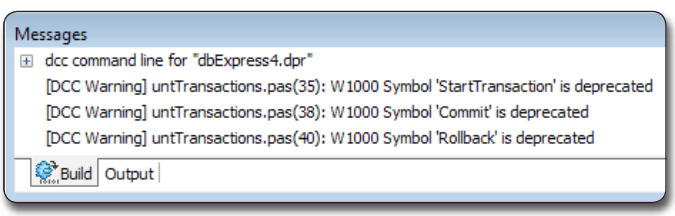


Figura 2. Mensagens do compilador indicando que determinados métodos estão obsoletos

Além dos novos métodos, o dbExpress 4 trouxe outros dois tipos de nível de isolamento que foram adicionados ao contexto de transações:

- Serializable: a transação corrente não visualiza quaisquer alterações realizadas por outras transações;
- SnapShot: caracteriza uma transação somente leitura que provê um snapshot (uma imagem fiel) transacional do banco de dados;

Apesar de substituídos, métodos antigos como StartTransaction, Commit e Rollback ainda existem nas versões recentes do Delphi, como por exemplo, a XE, por questões de compatibilidade. Contudo, ao se utilizar algum deles o compilador emite um aviso de que determinado método está "deprecated", ou seja, obsoleto, conforme pode ser visto na **Figura 2**.

DataSnap

Anteriormente quando se falava em desenvolvimento multicamadas DataSnap no Delphi não se fazia qualquer tipo de menção ao dbExpress, visto que ambas tecnologias possuíam objetivos distintos. No entanto a partir do dbExpress 4, mais precisamente com o lançamento do Delphi 2009, isso mudou. Isto porque o framework DataSnap foi também totalmente remodelado, sendo que uma de suas novidades foi a integração com um novo driver cliente dbExpress exclusivo para esta arquitetura. Explicando melhor: desde então, numa solução DataSnap as aplicações clientes poderiam facilmente se conectar a um Servidor de Aplicações apenas utilizando um componente TSQLConnection que faça uso do então novo driver DataSnap. Esta "simples" mudança no desenvolvimento fez com que esse tipo de solução se tornasse um dos atrativos nas versões seguintes do IDE, justificado pela facilidade proporcionada ao desenvolvedor Delphi. Tanto que até mesmo programadores menos experientes que tinham as aplicações

multicamadas como um "bicho de sete cabeças", se encorajaram a direcionar suas atividades para DataSnap.

Um novo jeito de definir conexões

Ainda no Delphi 7 (somente para tomar como exemplo), ao se utilizar os componentes VCL dbExpress para a parte de conexão com banco de dados de uma aplicação, a primeira atitude a se tomar era definir a conexão em si através de um TSQLConnection. A maneira mais fácil e tradicional para definir os parâmetros desta conexão era selecionando seu editor de propriedades (opção Edit Connection Properties), que era conseguido com um duplo-clique sobre o componente. Uma nova caixa de diálogo se abria, conforme pode ser visto na **Figura 3**. Nela, estavam disponíveis todos os drivers disponíveis, possibilitando a definição de uma conexão, bem como seus parâmetros. Todavia, esta forma de trabalho foi alterada no dbExpress 4. A partir do Delphi 2009 a opção Edit Connection Properties deixou de existir. Desde esta época, a maneira mais fácil para se definir uma nova conexão dbExpress é utilizando o Database Explorer.

O Database Explorer, para quem não conhece, é um aplicativo que acompanha a instalação do Delphi (pode ser encontrado em sua pasta Bin), e tem a incumbência de gerir conexões com bancos de dados. Uma de suas principais vantagens é possibilitar, por exemplo, a manipulação dos objetos do banco, tais como: tables, views, procedures e functions. Ele se integra ao IDE do Delphi em forma de janela, possibilitando trabalhar com código e banco de dados num ambiente único, sem a necessidade de ferramentas adicionais.

Você pode, só para citar, abrir e visualizar os dados e metadados de uma tabela ou ainda executar comandos SQL de dentro da própria IDE, em tempo de design. Até o Delphi 2005 ela era totalmente direcionada ao BDP para ADO.NET, em função do suporte a projetos ASP.NET do IDE, que era um dos atrativos na ocasião. Já no Delphi 2006 foi adicionado o suporte a conexões dbExpress no Database Explorer, obviamente voltado para as aplicações Win32, que era e continua sendo o seu carro-chefe. Com o lançamento do então remodelado dbExpress 4, no Delphi 2007, o suporte a BDP no Database Explorer deixou de existir e o dbExpress tornou-se

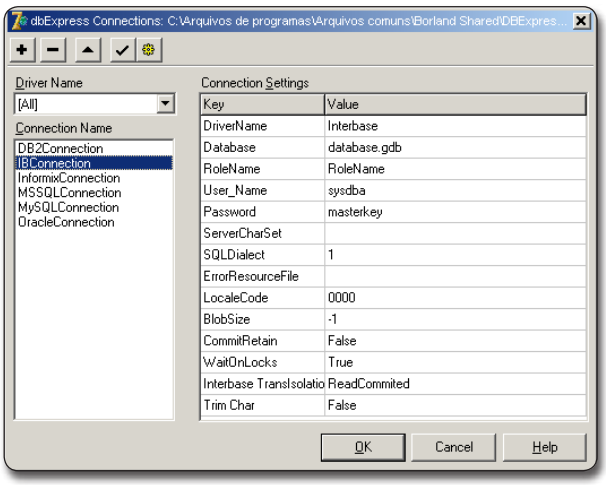


Figura 3. Definindo uma conexão dbExpress no Delphi 7

Nota do DevMan

OLEDB (Object Linking and Embedding Database) pode ser definido como um conjunto de componentes de software, criados pela Microsoft, que proporcionam a comunicação entre aplicação e banco de dados, tais como Oracle, SQL Server, MySQL e DB2. É muito comparado ao ODBC e em alguns casos, interpretado como seu sucessor devido ao seu melhor desempenho.

Nota do DevMan

TTransIsolationLevel - Tipos de nível de isolamento para transações dbExpress até a versão 3:

- xilREADCOMMITTED: a transação corrente só visualiza as mudanças que foram devidamente efetivadas (comitadas);

- xilDIRTYREAD: a transação corrente visualiza todas as alterações, inclusive as realizadas por outras transações, mesmo que não tenham sido efetivadas;

- xilREPEATABLEREAD: a transação atual garante uma visualização consistente dos dados, incluindo somente as alterações de outras transações efetivadas no início da transação;

- xilCUSTOM: a transação atual utiliza um nível de isolamento específico para determinado banco de dados.

Nota do DevMan

JSON é a sigla de JavaScript Object Notation. É uma notação baseada em texto de modo a representar objetos JavaScript que possam ser persistidos ou transferidos de uma aplicação para outra. Muito comparado ao XML, porém popularmente conhecido por ser mais compacto e de mais fácil compreensão.

REST é o acrônimo de Representational State Transfer (Transferência de Estado Representacional). Pode ser definida como uma forma de projetar aplicativos Web fracamente acoplados, cujos recursos são nomeados e acessíveis diretamente através de uma URI (Identificador Uniforme de Recursos) exclusiva para cada recurso.

Listagem 5. Exemplo simples da utilização de um tipo JSON

```
01 var
02   jArray:TJSONArray; //dar uses em DBXJSON
03 begin
04   jArray := TJSONArray.Create;
05   jArray.Add(1);
06   jArray.Add(2);
07   jArray.Add(3);
08   ShowMessage('Elementos de jArray:' + jArray.
     ToString);
09 end;
```

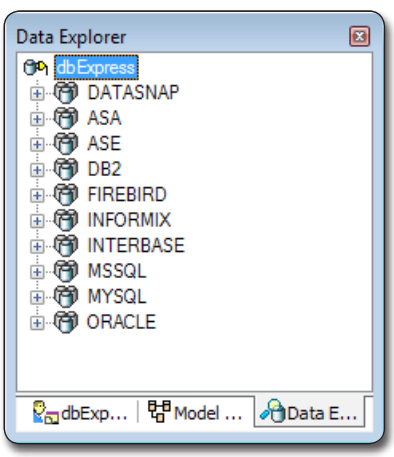


Figura 4. Database Explorer

único e exclusivo, conforme pode ser visto na **Figura 4**. O simples arrastar de uma das conexões registradas para um formulário de um projeto faz com que seja adicionado automaticamente um componente TSQLConnection, da paleta dbExpress, já configurado para a conexão. O mesmo acontece com os demais objetos, como no caso das tabelas, onde um TSQLDataSet é adicionado já direcionado à tabela e à conexão referente.

Com o surgimento do novo driver dbExpress para DataSnap, foi adicionado ao Database Explorer o suporte a este tipo de conexão. Em outras palavras, agora é possível, além das conexões com banco de dados se registrar uma conexão com um Servidor de Aplicação DataSnap.

Nova unit DBXJSON

O suporte a JSON no Delphi se deu ainda no passado através da utilização de algumas bibliotecas específicas de terceiros. Todavia com o lançamento do Delphi 2010, o suporte a JSON tornou-se nativo através de uma série de classes definidas na nova unit intitulada DBXJSON, parte integrante do framework dbExpress 4. O JSON ganhou ainda mais importância no contexto Delphi, visto também o novo suporte a REST, em servidores DataSnap. Dessa forma, estes tipos de servidores tornaram-se capazes de retornar seus clientes, valores no formato JSON, permitindo que esses fossem lidos facilmente também por outras linguagens.

As classes de DBXJSON permitem a manipulação de diversos tipos de dados JSON, tais como: tipos de valores primitivos (TJSONTrue, TJSONFalse), estrutura de dados (TJSONObject), pares (TJSONPair) e matrizes (TJSONArray). A seguir, na **Listagem 5** vê-se um exemplo simples da utilização de uma das inúmeras classes JSON disponibilizadas pela unit DBXJSON.

Ainda relacionado à DataSnap, a unit DBXJSON traz consigo uma classe denominada TDBXCallback, que é a classe responsável por implementar este tipo de funcionalidade em aplicações distribuídas. Para melhor entender, o conceito de Callback é baseado em uma requisição cliente ao Servidor de Aplicações, que envia respostas a esta requisição de forma síncrona. Todo o tráfego envolvido entre as camadas faz uso de objetos do tipo JSON do Delphi, para a troca de informações.

Conclusão

Acompanhamos neste artigo toda a evolução do dbExpress ao longo dos lançamentos de cada versão do Delphi. Sua versão atual, de número 4, mostra o resultado de toda a maturidade que o framework atingiu durante essa jornada. Versão essa que teve seu início ainda no Delphi 2007, onde a arquitetura teve suas atualizações mais significativas.

O framework evoluiu a ponto de se atrelar a outras tecnologias presentes no Delphi. Um exemplo: agora é impossível você não pensar em dbExpress, ao se utilizar a tecnologia DataSnap para desenvolvimento multicamadas, como foi visto neste artigo.

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 5 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas além de participar do programa Beta Tester Delphi Prism, da Embarcadero.

Embarcadero RAD Studio XE Architect
https://downloads.embarcadero.com/free/rad_studio

Firebird 2.5
http://www.firebirdsql.org/index.php?op=files&id=engine_250

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

FÓRUM DEVMEDIA

O lugar perfeito para você ficar por dentro de tudo o que acontece nas tecnologias do mercado atual

No novo Fórum da DevMedia você vai encontrar canais específicos de Delphi, ASP.NET, Java, Banco de Dados e Engenharia de Software; além de ter contato com profissionais qualificados da área para troca de informações, sugestões e muito mais.

ACESSE AGORA
www.devmedia.com.br/forum

DevMedia

Delphi Prism

Melhorias da linguagem – Par

O Delphi Prism possibilita que um método seja definido de maneira assíncrona. Ao executar a chamada a um método assíncrono, sua chamada é retornada imediatamente e seu método não poderá retornar um valor ou aguardar por outros parâmetros. Para sua utilização basta adicionar a palavra chave async na definição do método, como pode ser observado na **Listagem 1** que representa um trecho de código.

Um método assíncrono permite que se utilizem técnicas de multi-threading em sistemas .NET executando mais de uma tarefa ao mesmo tempo sem que para isso seja necessário criar classes descendentes de TThread, ou seja, a maneira tradicional utilizada em outras versões do Delphi. Esta é uma forma simples de declarar métodos assíncronos, dessa forma, com a sua declaração como assíncrona, isto define que, em algum momento, a declaração conterá um determinado valor, que será calculado em um segmento separado. Se outro segmento tentar acessar o resultado de quem realizou a chamada assíncrona, ficará aguardando enquanto não obtiver o resultado para uso.

Chamadas assíncronas apresentam vantagem na utilização em aplicativos web, pois dessa forma, o aplicativo não parece estar congelado, enquanto aguarda o retorno de algum método. Obviamente se o método é chamado em resposta a algum evento, como um botão, o aplicativo terá que aguardar a resposta.

Ao implementar o método, seu código de exemplo está descrito na **Listagem 2.** A implementação possibilita utilizar este método de forma assíncrona podendo realizar outras operações no aplicativo sendo executadas em paralelo com este método. Quando terminar sua execução, será apresentada na tela uma mensagem, conforme sua definição.

Resumo DevMan

De que trata o artigo:

Este artigo irá destacar o que há de novo ou aperfeiçoado na utilização da linguagem do Delphi Prism, indicando alguns dos principais recursos que apresentam inovações ou funcionalidades adicionais relacionadas à linguagem e que não estão presentes em outras versões do Delphi.

Para que serve:

A utilização das melhorias de linguagem auxilia na tomada de decisões para a codificação de um sistema permitindo adicionar recursos de linguagem bastante eficazes e que também tornarão o desenvolvimento de código dos aplicativos mais conciso e transparente.

Em que situação o tema é útil:

Este tema visa ampliar os conhecimentos na utilização da linguagem. Dessa forma, possibilitando a criação de aplicativos com códigos aperfeiçoados e mais eficientes, permitindo um ganho de desempenho relacionado à codificação se comparado a outras versões do Delphi, pois apresentará construções de código mais simples.

Delphi Prism – Melhorias da linguagem :

No decorrer deste artigo será destacada a palavra Async que possibilita a definição de métodos de forma assíncrona de execução. O uso de Sealed para restringir características de herança, Partial na definição de códigos parciais permitindo realizar a divisão do código de uma estrutura em mais de uma parte. A adição de Params possibilitando utilizar a passagem de um número variável de argumentos em métodos. A determinação da avaliação de certas condições de restrição em expressões através do uso de Implies. Os bloqueios em métodos para garantir a persistência de código indicados pela palavra Locked. A utilização de Out na passagem de argumentos de um método por referência. O surgimento de Enum e Flags para criar enumerações, tipos enumerados ou coleções de itens normalmente com identificadores numéricos. As melhorias para estruturas de repetição adicionadas com o uso de Index na determinação do índice da estrutura e o Step como forma de incremento da mesma. O surgimento de Matching auxiliando a programação na filtragem e indicação correta de itens a serem mostrados descartando o que não pode ser apresentado. E por último a já conhecida palavra With, mas ampliada com novas funcionalidades e recursos que facilitarão ainda mais a programação.

Operações assíncronas representam uma característica presente no Delphi Prism, tornando possível a execução de uma ou mais declarações de métodos de maneira assíncrona na sua definição. Se as declarações forem realizadas desta forma, o .NET Framework saberá interpretar e adicionar a tarefa no seu pool de threads a serem executadas.

Uma variação do código apresentado anteriormente é indicada na **Listagem 3** onde a palavra reservada async é adicionada diretamente na estrutura de repetição for. O código representa as mesmas funcionalidades do método definido na **Listagem 2**, mas na sua definição de método, não estará presente a palavra async, e ela foi adicionada diretamente à estrutura de re-

Rodrigo Araujo

petição for, o que irá indicar que a estrutura é que será realizada em paralelo. Observe que ao executar a chamada ao método, a mensagem de retorno ao usuário em tela será observada de maneira diferente caso não fosse definida a palavra async, pois o método não ficará aguardando a estrutura de repetição terminar a sua execução.

Sealed

A palavra chave Sealed é utilizada para restringir características de herança, impedindo que uma determinada classe seja herdada. A **Listagem 4** indica a definição de uma classe utilizando Sealed. A definição da classe Pessoa indica duas propriedades, uma para o nome da pessoa e outra o seu telefone, como se fosse um cadastro de agenda de contatos.

O código da **Listagem 5**, que não é permitido, define uma nova classe usando herança da classe Pessoa. A definição da classe Pessoa indica que a utilização de Sealed restringe seu uso por outras classes, como no caso, a classe Pessoa2. Ao compilar o código, será gerado um erro em tempo de compilação informando desta restrição.

Partial

É uma palavra utilizada para a definição de um código parcial, ou seja, possibilita a divisão em mais de uma parte de código utilizando a mesma estrutura de classe. A plataforma .NET permite separar em arquivos distintos atributos e funcionalidades de uma mesma classe, assim, ao utilizar partial é facilitada a sua visualização e manutenção de código.

Na **Listagem 6** é indicada a definição de uma classe parcial. Foi definida uma classe parcial chamada MinhaClasse, dessa forma irá possibilitar que outras definições de classes parciais semelhantes utilizem suas características e possibilitem ampliar suas atribuições. Essa característica é demonstrada na **Listagem 7** que é utilizada em outro arquivo de código.

Apesar de serem arquivos diferentes, o uso da classe MinhaClasse, devido a sua definição como classe parcial, possibilitará utilizar os atributos presentes nas duas definições como se fossem definidos em apenas uma classe. A **Listagem 8** demonstra a sua utilização. Ao utilizar a classe parcial MinhaClasse, observe que os atributos nome e telefone estão presentes em definições parciais diferentes, e, além disso, em mais de um arquivo. Ao realizar a referência à classe, é possível utilizar os atributos definidos separadamente. Veja que esta é uma das grandes facilidades de utilização de classes parciais.

Além disso, a palavra chave partial também pode ser utilizada em métodos, permitindo, assim, a sua chamada, e, também a sua

Listagem 1. Código para demonstrar o uso de async

```
method testemensagem(mensagem: string; contador: Integer);  async;
```

Listagem 2. Código utilizado para demonstrar as funcionalidades do método

```
method MainForm.testemensagem(mensagem: string; contador: Integer);
begin
  for i:Integer:=1 to contador do
    mensagem := mensagem + i.ToString();
    MessageBox.Show(mensagem);
end;
```

Listagem 3. Variação de código para demonstrar o uso de async

```
method MainForm.testemensagem(mensagem: string; contador: Integer);
begin
  async..for i:Integer:=1 to contador do
        mensagem := mensagem + i.ToString();

    MessageBox.Show(mensagem);
end;
```

Listagem 4. Código utilizando Sealed na definição de uma classe

```
type
 Pessoa = sealed class
 public
  constructor; empty;
  property Nome: string;
  property Telefone: String;
 end;
```

Listagem 5. Código utilizando herança da classe Pessoa

```
type Pessoa2 = class(Pessoa) // erro, pois Pessoa é Sealed
private
public
end;
```

Listagem 6. Definição de uma classe parcial

```
type
 MinhaClasse = partial class
 public
  property Valor: Integer;
  property Nome: string;
 end;
```

Listagem 7. Definição da classe parcial e outros atributos

```
type
 MinhaClasse = partial class
 public
  property Idade: Integer;
  property Telefone: String;
 end;
```

Listagem 8. Utilização da classe definida anteriormente

```
var teste := new MinhaClasse();
teste.Nome := "Rodrigo";
teste.Telefone := "(55) 1111-1111";
```

Listagem 9. Definição de métodos parciais

```
type
 MinhaClasse = partial class
 public
  property Idade: Integer;
  property Telefone: String;
  method InserePessoa; partial; empty;
  method BuscaPessoa; partial;
 end;
```

Listagem 10. Utilizando params na definição de um método

```
method MainForm.SomaNumeros
(params numeros: array of Int32);
begin
  var i, total: Int32;
  total := 0;
  for each i in numeros do
  begin
    total := total + i;
  end;
  MessageBox.Show("Somatório é" + total.ToString());
end;
```

Listagem 11. Executando a chamada do método

```
method MainForm.button1_Click(sender: System.
Object; e: System.EventArgs);
begin
 SomaNumeros(3, 4, 5);
 SomaNumeros(3, 4, 5, 6, 7, 10);
end;
```

Listagem 12. Demonstração do uso de implies

```
method MainForm.DivideNumeros(PermiteDivisao:
Boolean; valor1: Int32; valor2 : Int32);
require valor2=0 implies PermiteDivisao;
begin
  if (valor2>0) then
    MessageBox.Show("A divisão é" +
     (valor1/valor2).ToString())
  else
    MessageBox.Show("Erro de cálculo!");
end;
```

opcional implementação. Um método parcial pode ser também definido utilizando a palavra empty em conjunto com a classe parcial definida, possibilitando "atrasar" a definição deste método ou até deixá-la em branco.

O código da **Listagem 9** demonstra a utilização de uma classe parcial em conjunto com a definição de métodos parciais. Um erro em tempo de compilação é indicado informando que o método BuscaPessoa não possui implementação, já ao observar o método InserePessoa não é apresentado nenhum erro devido ao uso de empty que possibilita realizar a definição de código posteriormente, inclusive se o método for utilizado, nenhum erro de execução será apresentado. Esta é uma boa funcionalidade para implementações futuras ou realizadas por mais de uma pessoa em uma equipe de desenvolvimento de sistemas.

Params

Surge para facilitar a definição de métodos que irão aceitar um número variável de argumentos. Ao utilizar params na definição de um método, este método apresentará um array de tamanho variável no número de elementos para utilização em suas chamadas. A **Listagem 10** apresenta a definição de um método utilizando params.

A chamada ao método SomaNumeros poderá ser realizada conforme a **Listagem 11**. É possível observar a chamada ao método com uma passagem variável de parâmetros, permitida devido ao uso de params na definição do método. Esta é uma situação que possibilita bastante flexibilidade para a execução de métodos.

Implies

A função deste operador é assegurar que determinadas condições sejam satisfeitas em algumas expressões, principalmente com o uso em conjunto de operadores require e ensure, que indicam situações de validação de expressões para dar prosseguimento à execução de um determinado código. Seu funcionamento é semelhante aos operadores de expressões booleanas como o and e o or, mas realizando uma implicação em todo o conjunto que está sendo avaliado, ou seja, a expressão caracterizada por implies somente será avaliada caso a expressão inicial for verdadeira, se a expressão inicial for falsa todo o conjunto de expressões será considerado válido.

Nota

Algumas versões do compilador Win32 também suportam esse recurso.

A **Listagem 12** demonstra a definição de um método que utiliza implies. A definição do método utiliza require para validar uma condição de entrada, neste caso o valor indicado pelo segundo parâmetro numérico representado pela variável valor2, que será utilizada no cálculo da divisão, e, não poderá ser o valor zero. A utilização de implies no método somente será avaliada se o valor indicado na variável valor2 seja zero, neste caso, então será avaliada a variável PermiteDivisao. Se o valor for diferente de zero, toda a expressão será considerada como verdadeira.

A **Listagem 13** indica quatro exemplos de uso do método definido. As situações são distintas: na primeira chamada é indicada a divisão e seu resultado o numeral 2, o segundo caso, mesmo apresentando a variável PermiteDivisao como false o resultado é o igual ao do primeiro, pois a variável valor2 é diferente de zero, dessa forma, toda a expressão é considerada como válida. O terceiro caso como a variável valor2 é indicada como zero, a condição indicada por implies é avaliada e retornando seu valor como false, assim, indicando uma mensagem na tela do usuário representada pela **Figura 1**. O último exemplo, mesmo sendo avaliada a expressão, retornará verdadeira devido ao uso de true como seu valor inicial.

Locked

Esta diretiva ao ser implicitamente declarada permite realizar bloqueios em métodos possibilitando assim uma persistência segura de código no segmento monitorado. Dessa forma o compilador irá gerar as chamadas adequadas de monitoração para o método locked que é utilizado principalmente em métodos não virtuais e opera no nível da classe, ou seja, um método irá bloquear ou travar a

execução de outro.

A monitoração é realizada diretamente pela classe monitor que controla o acesso aos objetos através da concessão de bloqueios por segmentos, assim possibilitando restringir o acesso a um determinado bloco de código, definindo uma sessão crítica. Além disso, quando uma thread possui o bloqueio para um objeto, nenhuma outra instância poderá adquirir este bloqueio até que seja liberado pela thread inicial, garantindo a persistência das informações.

A **Listagem 14** apresenta o uso de locked na definição de um método. Ao definir o método utilizando a palavra reservada locked é possível criar um bloco de código crítico para este método, sendo desta forma thread-safe, auxiliando na execução do código, obtenção e liberação de recursos alocados, descrevendo de maneira simples um modelo de programação multi-thread.

Out

A palavra reservada out possibilita a passagem de argumentos de um método por referência, sem precisar ser inicializada. A **Listagem 15** demonstra a utilização de out. Foi definido um método chamado SubtraiNumeros, responsável pela subtração de dois números e definição de uma variável x utilizando out e armazenando o resultado da subtração.

Já na **Listagem 16** é indicado o uso do método recentemente definido. É possível verificar a chamada ao método SubtraiNumeros e a utilização de uma variável chamada de resultado do tipo Int32, nesta variável estará presente após o cálculo o resultado da subtração do numeral 10 com o 7, e o resultado armazenado na variável como sendo o numeral 3 e apresentado na forma de mensagem na tela do usuário.

Enum

É utilizado para definir um tipo enumerado, que será usado na atribuição de identificadores numéricos para determinados valores. A **Listagem 17** demonstra a criação de uma enumeração de cores simples. Neste exemplo o compilador irá atribuir valores numéricos para cada um dos itens da enumeração, e por padrão, será atribuído o valor zero ao primeiro item, seguindo uma sequência incremental para cada item, ou seja, 0, 1 e 2.

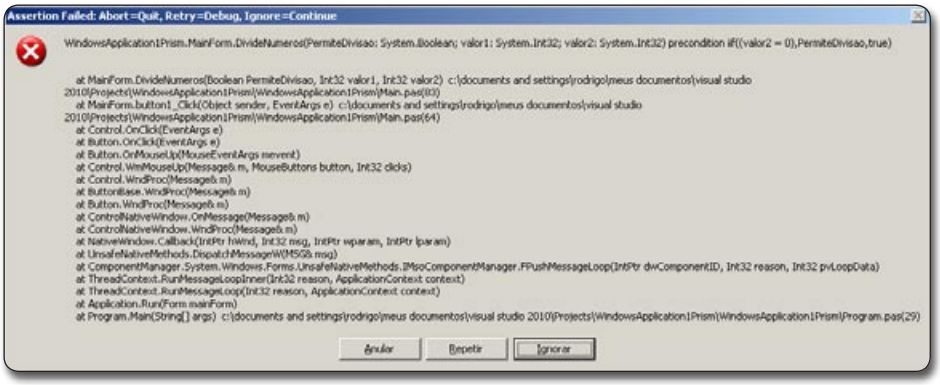


Figura 1. Restrição indicada pelo uso de Implies na pré-condição

De maneira semelhante na **Listagem 18** é possível verificar uma forma de modificar esta atribuição inicial e indicar outros valores a cada um dos itens. A enumeração estará relacionada a valores numéricos, caso haja necessidade de modificar o tipo padrão inteiro por outro, é possível realizar esta alteração conforme apresenta a **Listagem 19** com a definição de outro tipo e ainda por cima com um padrão de valores adicionados.

Flags

Possui um tratamento bastante semelhante ao que ocorre com enum, mencionado anteriormente, mas ao invés de inicializar a contagem em zero, apresenta seu início no numeral 1 e seus itens seguintes, por padrão, serão representados pela potência de dois, semelhantes a: 1, 2, 4, 8, 16 e assim por diante. A **Listagem 20** demonstra o uso de flags na indicação de valores. Observe que a definição é simples e de maneira análoga ao que ocorre com enum, diferenciando-se pelo índice inicial de valores, neste caso o numeral um e a potenciação dos valores subsequentes.

Da mesma forma, a **Listagem 21** demonstra uma variação desta definição e atribuição de valores diferentes do padrão. É possível verificar que o tipo EnumeraValoresAlterado apresenta uma alteração de valores, fugindo a regra padrão de definição, além disso, também é permitida a alteração do tipo, possibilitando

Listagem 13. Exemplificação do uso de implies

```
method MainForm.button1_Click(sender: System.
Object; e: System.EventArgs);
begin
  DivideNumeros(true, 4, 2);
  DivideNumeros(false, 4, 2);
  DivideNumeros(false, 1, 0);
  DivideNumeros(true, 1, 0);
end;
```

Listagem 14. Uso de locked na definição de um método

```
method MultiplicaNumeros(valor1: Int32; valor2:
Int32); locked;
```

Listagem 15. Definindo o uso de out em um método

```
method MainForm.SubtraiNumeros(out x: Int32;
valor1 : Int32; valor2 : Int32);
begin
  x := valor1 - valor2;
end;
```

Listagem 16. Realizando uma subtração para demonstrar o método

```
method MainForm.button2_Click(sender: System.
Object; e: System.EventArgs);
begin
  var resultado : Int32;
  SubtraiNumeros(out resultado, 10, 7);
  MessageBox.Show("O resultado da subtração é "
+ resultado.ToString());
end;
```

Listagem 17. Criando um tipo enumerado

```
type
  EnumeraCores = enum(Vermelho, Verde, Azul);
```

Listagem 18. Definindo um tipo enumerado e com definição de valores

```
type
  EnumeraCoresModificado = enum(Vermelho=10,
Verde=20, Azul=30);
```

Listagem 19. Modificação de um tipo enumerado e com valores agregados

```
type
  EnumeraCoresTipoModificado =
enum(Vermelho=10, Verde=20, Azul=30) of word;
```

Listagem 20. Utilizando flags na definição de um tipo

```
type
 EnumeraValores = flags(Valor1, Valor2, Valor3);
```

Listagem 21. Definindo uma alteração de valores com o uso de flags

```
type
 EnumeraValoresAlterado = flags(Valor1=5, Valor2,
 Valor3=10);
```

Listagem 22. Criando uma estrutura de repetição e utilizando index

```
method MainForm.SomaNumeros(tipo: Boolean;
valor: Int32; params numeros: array of Int32);
begin
 var i, total, x: Int32;
 total := 0;
 for each i in numeros index x do
 begin
  total := total + i;
  MessageBox.Show("Realizando iteração nº" +
  x.ToString());
 end;
 MessageBox.Show("Somatório é" + total.ToString());
end;
```

Listagem 23. Utilizando Step em uma estrutura de repetição

```
method MainForm.MostraImpares();
begin
 for i: Int32 := 1 to 365 step 2 do
 begin
  MessageBox.Show(i.ToString());
 end;
end;
```

Listagem 24. Uso de Matching em uma estrutura de repetição

```
method MainForm.button3_Click(sender: System.
Object; e: System.EventArgs);
var
 x: ArrayList;
begin
 x := new ArrayList();
 x.add('Teste de escrita');
 x.add(123);
 for each matching S: String in x do
  MessageBox.Show(s);
end;
```

Listagem 25. Uso de With inferindo o tipo de uso

```
method MainForm.button2_Click(sender: System.
Object; e: System.EventArgs);
begin
 with l: Label := new Label do
 with c: Color := l.BackColor do begin
  l.Text := 'Hello World';
  l.Left :=  10;
  l.Width := 100;
  self.Text := c.ToString;
 end;
end;
```

definir outros tipos de uso que não sejam o padrão, semelhante ao que ocorre na **Listagem 19**.

Index

A palavra reservada "index" é utilizada em conjunto com uma estrutura de repetição do tipo for each e é a palavra responsável por indicar qual é o índice numérico que está sendo executado no momento. Sua utilização é opcional e seu índice inicia a contagem incremental de zero até o fim da execução da estrutura de repetição, ou seja, o número de elementos presentes. Apresenta uma utilidade simples onde na execução ou ao final dela é interessante saber o número de elementos utilizados ou processados pela estrutura de repetição.

A **Listagem 22** demonstra o uso de índex em uma estrutura for each. Ao observar o código descrito anteriormente é verificado que uma mensagem na tela será apresentada a cada etapa de execução da estrutura de repetição. Uma maneira interessante de utilizar index é relacioná-lo a um datagridview ou um dataset onde cada valor numérico representado nesta estrutura seria considerado como um registro contendo campos a serem utilizados em um aplicativo.

Step

É uma palavra reservada utilizada em conjunto com outras em uma estrutura de repetição. Step é apresentado para restringir o processamento de apenas uma parcela do conjunto de valores que serão utilizados ou examinados, sua indicação determina o valor a ser usado como incremento e sua definição não influenciará nos valores presentes na estrutura de repetição.

A **Listagem 23** demonstra o uso de step em um trecho de código. A definição do método indicará o uso de step com incremento de dois números informando que ao iniciar a variável inteira i com o valor numérico um, será apresentada na tela, na forma de mensagem, os números ímpares presentes no intervalo até o numeral trezentos e sessenta e cinco, ou seja, 1, 3, 5, 7 e assim por diante.

Matching

A palavra reservada matching é utilizada para facilitar a programação onde seu uso possibilita que sejam filtrados os dados corretos a serem apresentados ignorando as possíveis exceções relacionadas a trechos de códigos. Em outras palavras matching irá apresentar os dados de tipos corretos descartando os que não poderão ser apresentados.

A **Listagem 24** indica o uso de matching em uma estrutura de repetição for. Observe que ao utilizar matching somente será mostrada em tela a mensagem contendo o texto 'teste de escrita', isto se deve ao fato do tipo ser definido como String e ao adicionar um valor numérico ele não será apresentado, pois foi descartado por matching. Caso não fosse utilizado matching, uma mensagem de erro em tempo de execução seria apresentada na tela.

With

Com seu uso já conhecido, para o Delphi Prism ele poderá ser utilizado de maneira estendida com novas características de utilização. Uma de suas características é a possibilidade de inferir o tipo a ser utilizado de acordo com o valor atribuído. Além disso, apresenta o uso de variáveis de forma aninhadas de maneira bastante clara e precisa. E, obviamente, apresenta toda a compatibilidade com suas funções básicas utilizadas em outras versões do Delphi.

A **Listagem 25** apresenta um exemplo do uso de with com o tipo inferido a ser utilizado. O código permite visualizar os tipos utilizados e algumas de suas características básicas, além de utilizar with de maneira aninhada, observando com clareza cada linha de código apresentada de forma bastante transparente, isto devido ao uso de variáveis distintas para cada palavra reservada with utilizada.

Como outra utilidade, with pode ser utilizado em conjunto com matching, anteriormente demonstrado, para facilitar a validação de informações. Na **Listagem 26** é possível verificar o uso desta funcionalidade. Aqui é utilizada uma classe chamada de ClassePessoa e os dados serão analisados e, se não estiverem de forma correta, serão

Nota do DevMan

Para quem irá escrever consultas LINQ, a parte mais visível da integração com o Delphi Prism fornecida pela LINQ são as expressões de consulta. Expressões de consulta são escritas com uma sintaxe declarativa que foi introduzida na versão 3.0 da linguagem C# e codificada para o Delphi Prism. A sintaxe de consulta permite realizar, com um mínimo de código, diversas operações em dados, como: filtragem, projeção, ordenação, agrupamento, junção e outras.

As mesmas expressões de consulta LINQ são usadas para consultar coleções de objetos na memória (LINQ to Objects), dados em bancos de dados relacionais (LINQ to SQL e LINQ to Entities), objetos DataSet na memória (LINQ to DataSet), documentos XML na memória (LINQ to XML) ou, ainda, qualquer outra fonte de dados que tenha um LINQ Provider disponível.

A adição do LINQ ao Delphi Prism possibilita ao desenvolvedor realizar consultas complexas de uma forma bem mais simples, permitindo que ele foque no resultado desejado sem se preocupar em como os dados estão sendo extraídos. O recurso permite usar um raciocínio similar ao fornecido pela linguagem SQL, em bancos de dados relacionais, para fazer consultas em fontes de dados variadas diretamente na linguagem do Delphi Prism.

Nota do DevMan

Em bancos de dados relacionais, o SQL permite criar apelidos para expressões na lista da cláusula SELECT e estes apelidos podem ser usados nas cláusulas ORDER BY. Porém, por uma limitação da linguagem, os apelidos não podem ser usados na cláusula GROUP BY. Neste último caso, somos obrigados a copiar uma expressão para usá-la novamente na lista de agrupamento, ao invés de usar somente o apelido. Certamente, a introdução de uma cláusula WITH no SQL, similar à usada nas expressões de consulta LINQ, seria muito útil.

descartados, ou seja, caso não possua informações a serem utilizadas (valores nulos) nenhum dado será apresentado.

Outra funcionalidade para with é sua utilização em expressões de consulta, juntamente com outras palavras reservadas e operadores. Dessa forma, irá auxiliar na definição de novas variáveis que possam ser utilizadas em consultas LINQ reduzindo assim a complexidade das consultas e sem ter que repetir informações ou valores já representados.

Listagem 26. Usando With em conjunto com matching

```
with matching p: ClassePessoa := ClassePessoa(buscaPessoa()) do retornaPessoa;
```

Listagem 27. Utilizando with em uma expressão de consulta

```
from arquivo in Directory.GetFiles('C:\Windows\System32')
with extensao:= Path.GetExtension(arquivo).ToUpper()
with nomearquivo := Path.GetFileName(arquivo) order by extensao desc , nomearquivo asc
select new class (a:=nomearquivo, b:=extensao);
```

A **Listagem 27** define o uso de with em uma consulta LINQ. A cláusula with permite introduzir uma variável para armazenar resultados de expressões intermediárias numa expressão de consulta. Deste modo, o resultado armazenado na variável poderá ser reutilizado na consulta. Além disso, foram criadas duas variáveis na consulta LINQ (nomeArquivo e extensao) com o objetivo de que os resultados das expressões sejam utilizados na cláusula order by para ordenação e na cláusula select para projeção.

Conclusão

O uso das melhorias de linguagem presente no Delphi Prism possibilita novas e mais enxutas maneiras de realizar a programação de sistemas. Auxiliam para melhorar o desenvolvimento e adicionar funcionalidades que até o momento ainda não eram apresentadas. Elas ampliam o conhecimento e permitem uma nova visão para a criação de códigos.

Se comparadas a outras versões do Delphi, elas ajudam a criar códigos mais eficientes e menores, pois na maioria dos casos muitas estruturas para serem igualmente representadas em versões anteriores precisariam de várias linhas de código, onde no Prism será acelerada a produção.

No decorrer do artigo foram vistas algumas palavras de linguagem como o uso de Async, Sealed, Partial, Params, Implies, Locked, Out, Enum e Flags, Index, Step, Matching e With auxiliando no tratamento mais enxuto e eficaz para a geração de códigos.

Assim, permitirão um desenvolvimento mais robusto, elegante e de maneira simples que irão proporcionar um ganho incremental para a equipe que utilizar.

Rodrigo Araujo
rodrigo.arsi@gmail.com
Bacharel em Ciência da Computação pela UFSM – Santa Maria/RS trabalha com Análise e Desenvolvimento de Software há mais de 10 anos. Desde 2003, utiliza tecnologias .NET.

Site oficial do Delphi Prism
http://www.embarcadero.com/products/delphi-prism

Wiki do Delphi Prism
http://prismwiki.codegear.com/en/Main_Page

Download do Delphi Prism
https://downloads.embarcadero.com/free/delphi_prism

Site oficial do ASP.NET
http://www.asp.net

Visual Studio
www.microsoft.com/visualstudio

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Biblioteca madColl

Muito além de exceptions e hooki

O próprio Delphi possui uma vasta biblioteca de funções com vários objetivos. Mesmo assim sempre há a necessidade de se fazer download de bibliotecas de funções na internet ou de se criar uma biblioteca pessoal que possa ser usada em vários projetos.

O objetivo deste artigo não é esmiuçar ou servir de documentação à coleção madShi, até porque o projeto já possui uma documentação que pode ser vista na seção de links. Esta documentação é muito grande, e mesmo assim não é completa levando-se em conta a quantidade de aplicações práticas que essas units podem ter, mas a granularidade dessas necessidades. O uso ou não da biblioteca madShi dependerá muito das necessidades do projeto. Haverá ocasiões em que a biblioteca madShi será a melhor opção, e haverá outras ocasiões onde a coleção não será a melhor saída, mas será a saída mais simples ou rápida.

O pacote madBasic contém oito units com funções de propósito geral. Dentre essas oito as mais úteis no dia-a-dia são a madZip para compressão, a madCrypt para criptografia, a madLists que possui a implementação de alguns tipos de listas que não tem nativamente no Delphi, e a madStrings que possui algumas funções para tratar strings que geralmente são feitas pelo próprio programador.

Se uma empresa já tem a coleção de pacotes madShi os programadores podem considerar essas funções em vez de procurar mais algum pacote de terceiros ou de tentar criar do zero. A seguir algumas das funções da biblioteca.

Compressão: tem o objetivo de eliminar o uso de componentes de compressão complicados para tarefas de compressão simples. A unit madZip fornece um compressor de strings fácil de usar quando o objetivo for simplesmente comprimir strings. Isso pode ser útil onde o problema é espaço e não processamento. Um exemplo é a transmissão, pela internet ou através de um dispositivo de armazenamento, um grande lote de informações que é exportado por um sistema e importado por outro no final do dia (batch).

Criptografia: sem usar componentes de criptografia de terceiros a biblioteca madShi oferece serviços de criptografia chamando-se apenas uma função. O algoritmo utilizado é o "BlowFish". Não é o mais seguro de todos, mas é livre de patentes.

Strings: uma série de funções de busca, quebra, formatação e comparação de strings.

Tools: funções para obter informações do sistema operacional, transformar um método em uma procedure estática ou uma procedure estática em método.

Os pacotes madShell e madTools oferecem funcionalidades para trabalhar com o Shell do Windows (Explorer), arquivos e listagens de diretórios especiais (diretórios cujo local pode mudar de acordo com a instalação, e que geralmente ficam gravados no registro) como o diretório Arquivos de Programas, Windows ou Meus Documentos.

Algumas funções do pacote madShell incluem obter o diretório especial do Windows ou o diretório do sistema. Ideal para usar em processos de instalação e configu-

Resumo DevMan

De que trata o artigo:

A coleção madShi é uma espécie de canivete suíço para Delphi cujas funções vão desde manipulação de strings à segurança. Ela pode rapidamente suprir uma série de necessidades básicas dos programadores sem tirar o foco do negócio.

Para que serve:

O objetivo principal da coleção madShi é suprir algumas "ausências" da API do Windows ou na VCL/RTL. Ela pode ser útil na automatização de tarefas e para explorar recursos pouco conhecidos ou pouco documentados do Windows.

Em que situação o tema é útil:

A coleção madShi pode ajudar uma empresa a tirar um pouco o foco do mundo "software" e focar mais no negócio. Isso porque a coleção terceiriza certos serviços, como compressão, criptografia e segurança transformando-os apenas em chamadas a funções estáticas.

Biblioteca madShi:

Muitos programadores baixam ou criam bibliotecas de uso geral e de fácil chamada para procurar ou formatar strings, encontrar um diretório compartilhado, criptografar uma senha entre outras funcionalidades. Algumas dessas funcionalidades podem ser supridas pela VCL e pela RTL, outras podem ser vistas na JCL do projeto JEDI e ainda existem várias bibliotecas de funções a serem citadas. A coleção de bibliotecas madShi possui oito pacotes, cada um tendo de duas a nove units. Os pacotes madExcept para tratamento de exceções e madCodeHook para API Hooking já foram tratados em artigos anteriores dessa revista. Este artigo visa tratar de alguns aspectos interessantes dos seis pacotes restantes.

ection
ng

Vitor Luiz Rubio

ração onde um sistema precisa identificar onde é o diretório de sistema do Windows para poder copiar uma dll ou bpl.

Existe também a função GetSpecialFolder que aceita como parâmetro um objeto da coleção TSpecialFolder. Se for passado, por exemplo, sfMyMusic, a função GetSpecialFolder trará o diretório Minhas Músicas do usuário e configuração correntes. A coleção TSpecialFolder tem mais de 60 membros, cada um trazendo um diretório especial diferente. Há neste pacote um conjunto de referências a procedures estáticas ou métodos de objetos feitos para se monitorar eventos do shell. Os eventos monitorados vão desde operações com arquivos (inclusão, exclusão, troca de nome) a mapeamento de unidades de rede.

Funções para listar ShellObjects são úteis para se trabalhar com objetos do sistema operacional que são "navegáveis", mas não são arquivos ou diretórios no sistema de arquivos. Por exemplo, o painel de controle. Embora ele pareça uma pasta em algum lugar do disco, na verdade ele é uma espécie de programa escondido em um módulo do sistema operacional.

A função OS permite identificar qual é o sistema operacional e versão antes que algum recurso seja usado. Por exemplo, no Windows 98 e ME não é possível ter janelas semi-transparentes com a API do Windows, então o sistema operacional pode ser verificado antes de se tentar alterar a propriedade alphablend de um formulário, ou de acessar um diretório que só existiria no Windows NT.

As funções GetFileVersion e FileVersionToStr são especialmente úteis na hora de se gerenciar download de componentes e atualização do sistema. A função GetLongFileName traz o nome longo de um arquivo quando passado como parâmetro um nome curto de 8 caracteres do DOS (com "~") e a GetShortFileName faz o contrário.

NewShortCut é uma maneira fácil de criar um atalho para qualquer programa ou arquivo sem precisar conhecer a fundo a API do Windows. Ao longo do artigo será mostrado o método para criar um atalho sem a NewShortCut e com a NewShortCut.

DisplayMode pode ser usado para listar as resoluções suportadas pelo driver de vídeo corrente e é especialmente útil em aplicações gráficas e games.

O pacote madSecurity pode ser dividido em Accounts, ACE, ACL, Security Objects e Shares. Não será possível estudar exemplos de todos os módulos. Dentre eles é possível citar:

As função Account retorna um objeto da interface IAccount que é uma conta de usuário do Windows. O método CurrentUser retorna o usuário corrente. A função Shares retorna os compartilhamentos (pastas, impressoras ...) de um servidor (uma máquina Windows qualquer da rede que tenha algum compartilhamento).

O pacote madKernel tem funções para se usar recursos não documentados do Windows que a própria API não oferece de maneira tão fácil. madRemote é o pacote necessário para criar processos que interfiram em outros através de comunicação de baixo nível, na verdade escrevendo código executável diretamente no contexto de outro processo. É uma técnica muito avançada e exige conhecimentos teóricos sobre a estrutura dos arquivos PE do Windows, chamadas a procedimentos e assembly. Um exemplo é a CopyFunction, que é usada internamente na coleção madCodeHook. Ela tem o poder de copiar uma função do programa que a chamou para dentro de um outro programa que esteja rodando. Usada em conjunto com a função CreateRemoteThread permite que uma função externa seja executada dentro de um programa já rodando no contexto (variáveis, endereço de memória) dele, e não no contexto de quem disparou a thread. Basicamente essas duas funções juntas fazem injeção e execução de código alheio em processos que já estão executando.

Algumas funções existentes na API do Windows XP, 2000, 2003, Vista e 7 não existem no Windows 9x ou ME, ou no Windows NT4. Muitas das funções da biblioteca madShi servem para equalizar essa discrepância, pois tapam alguns buracos no Windows 9x, ME e NT4 justamente substituindo, e simulando muito precisamente, essas APIs que eles não têm. É o caso da EnumProcesses, que pode facilmente listar os processos em execução em qualquer SO Windows.

As funções do Windows para lidar com processos, seja por criá-los, ou seja, por enumerá-los às vezes, trabalham com o handle (manipulador) do processo, às vezes com o ID. A função ProcessHandleToId serve para descobrir o ID de um processo dado o seu handle.

A madDisasm simplesmente permite obtenção ou modificação do código assembly do programa sendo executado em tempo real. Útil quando você tem uma biblioteca de terceiros sem fontes de uma versão antiga do Delphi. Com base no assembly de algumas funções que não se tem o source é possível entender como o método funciona e até "clonar" o código numa versão mais moderna do Delphi.

Não é possível tratar cada uma das funções da biblioteca, pois ela é muito vasta, mesmo sendo menor que a VCL/RTL ou mesmo a JEDI. A coleção madShi tem algumas funções que já são conhecidas de quem usa JEDI ou DevExpress, mesmo assim tem algumas funções inéditas que facilitam bastante o trabalho do dia-a-dia, principalmente no que tange à integração com o Windows.

Muitas das funções servem para tornar disponíveis processos e recursos de baixo nível como se fossem nativos do Windows ou normalizar as deficiências da API do Windows em versões mais antigas a fim de simular a API dos Windows mais modernos.

Sendo assim é possível fazer com que um programa desenvolvido para Windows XP ou 7 que normalmente não rodaria em um Windows NT4/98 pela substituição de certas chamadas a funções da API por chamadas às funções correspondentes da madShi.

Mesmo tendo suas utilidades próprias a maioria das outras bibliotecas madShi serve de base, ou alicerce, para a madCodeHook e madExcept, já vistos num artigo anterior. Pode parecer que não há nenhuma necessidade específica para essas funções. Na maioria dos casos em que aparecer a necessidade de usá-las esta necessidade estará escondida em um requisito técnico em vez de uma regra de negócio.

Um exemplo claro disso é quando o cliente pede que um determinado arquivo seja salvo na pasta Meus Documentos e que um log de envio seja salvo no Desktop. Meus Documentos está na verdade, geralmente, na pasta "c:\documents and settings\<Nome do Usuario>\documentos". E a desktop é análoga: "c:\documents and settings\<Nome do Usuario>\desktop". Isto falando-se em Windows XP, pois no Windows 7 o caminho é diferente. O maior desafio aqui é obter o nome do usuário logado. Poderá haver, e geralmente há, mais regras de negócio e lugares no computador que são dependentes de usuário.

Pode ser também que o cliente peça que o sistema suprima sua própria tela de login, pois será usado o logon do Windows e as configurações de acesso do Active Directory para fazer login no sistema. Logo, em vez de o sistema abrir uma janela de login deverá identificar o usuário logado, e baseado em uma tabela interna saber o que ele deve ou não ter direito de fazer. O sistema deve manter essa tabela de usuários copiando-a do Windows e não sendo mantida por um usuário administrador como é o costume. Pode ser também um misto de ambos. A questão é: para que o usuário deveria digitar seu login e senha duas vezes? O Windows já provê uma política de segurança, com diretivas de mudança de senha periodicamente e restrições de acesso muito mais maduro do que qualquer sistema no início de sua vida útil. Além disso, uma mudança na senha do Windows causaria uma diferença entre a senha do computador/rede e a do sistema, o que é motivo de confusão para usuários mais leigos.

Evitar impor certas cargas aos usuários, como saber corretamente onde salvou um documento ou decorar muitas senhas pode trazer um pequeno ganho de produtividade e problemas a menos para o pessoal de suporte.

Um Web Server em uma intranet, por exemplo, o IIS, pede a senha do Windows para saber se o usuário pode ou não acessar a página da Intranet. Dependendo das configurações o IE pode verificar que o usuário já está logado com uma conta que é autêntica e tem permissão de acesso e simplesmente deixa o usuário passar. Outros programas do Windows também fazem a mesma coisa.

O SharePoint é um sistema da Microsoft muito poderoso, para criação de sites, intranets, fóruns, blogs, base de conhecimento, CMS e gerenciamento de documentos. Para acesso ao SharePoint ele permite um misto entre autenticação pelo Windows ou autenticação pelo sistema interno dele.

Delegar a autenticação de um usuário por confiar que ele já esteja autenticado por uma outra autoridade é o que muitos sites têm feito, substituindo seus sistemas internos de autenticação pelo Twitter, Facebook ou OpenID. O uso ou não dessas técnicas dependerá dos requisitos não funcionais referentes à segurança do sistema.

Existe ainda a possibilidade de se fazer o contrário: substituir a tela de logon nativa do Windows pela tela de logon do sistema ou ERP da empresa.

Embora o Delphi já proporcione muitas dessas funcionalidades e algumas funções pareçam duplicidade de código, Mathias Rauen, autor da biblioteca madShi, garante que as funções dele são extremamente rápidas. Isso porque ele usou várias regras e lógicas de otimização, e em muitos casos usou até assembly para fazer certas funções. Então um bom lugar para se usar qualquer função da madCollection é dentro de um loop muito grande onde já são usados os métodos nativos do Delphi. Substituindo-os por métodos semelhantes da mad pode aumentar um pouco a performance do sistema em certos lugares.

Para criar esse artigo nenhum sistema real foi criado para que o leitor possa verificar a independência de contexto das funções aqui apresentadas e reconheça por si as melhores ocasiões onde essas bibliotecas possam ser usadas.

Juntamente com esta edição pode ser feito o download dos arquivos de exemplos. Todos os exemplos foram feitos em um formulário com vários botões e a saída de todos os testes é mostrada em um memo. A **Figura 1** mostra como pode ser esse programa de teste, que não tem nenhuma utilidade senão apresentar a madShi.

Comumente, em testes de bibliotecas, usa-se uma ferramenta de testes unitários, principalmente se o fornecedor da biblioteca fornece também os testes prontos, e, a fim de fazer um "tutorial" para a equipe de trabalho, usualmente se faz um teste em um formulário, com vários botões para testar as várias funcionalidades, oferecendo assim um ambiente um pouco mais "real".

Um exemplo dessa prática pode ser encontrado em pacotes de componentes como JEDI, DevExpress e VCL Skin, onde o teste nada mais é que um exemplo de formulário onde os componentes e alguns de seus eventos e propriedades são utilizados.

Neste artigo as funções serão explicadas fora da ordem em que aparecem na biblioteca. Algumas bibliotecas foram divididas em mais de um subtítulo outras foram agrupadas debaixo do mesmo. A seguir temos uma lista sucinta das units e as funções usadas neste artigo que podem ser encontradas em cada uma delas:

- madStrings - funções para comparação, busca, exclusão, manipulação e conversão de strings;
- madDisAsm - ParseFunction, ParseCode;
- madZip - Compress, Uncompress, Zip;
- madTypes - tipos como o TDAUnicode String;
- madCrypt - Encrypt, Decrypt, Encode, Decode;
- madTools - OS, MethodToProcedure, GetLongFileName, GetShortFileName;
- madShell - NewShortcut, GetSpecialFolder, RegisterShellEvent;
- madSecurity - account, shares;
- madRemote – EnumProcesses.

Esta lista contém apenas as units usadas neste artigo. Da madRemote apenas EnumProcesses foi usada, pois todas as outras funções desta unit são para hacks avançados como copiar uma função para dentro de outro processo rodando, criar uma thread em um processo rodando e outras coisas que agradariam apenas estudantes de segurança.

Strings

String é o tipo de dado mais importante quando se fala de sistemas comerciais ou de negócios. São cadeias de caracteres que podem significar qualquer coisa, dependendo do contexto.

As funções CompareText e CompareStr funcionam perfeitamente para comparações entre palavras com caracteres especiais sem a necessidade de se chamar a função AnsiCompareText. Outro motivo de se usar as funções da biblioteca madShi é que elas são rápidas.

Além disso, se um programa do tipo console precisa ser pequeno, convém usar as funções da madCollection em vez da AnsiCompareText, porque a AnsiCompareText está dentro da unit SysUtils, e apenas adicioná-la ao projeto pode fazer o tamanho do executável aumentar consideravelmente.

Estas funções retornam zero se as strings forem consideradas iguais, um número positivo se a primeira é lexicograficamente maior que a segunda e um número negativo se a primeira é lexicograficamente menor que a segunda. O padrão ou ordem lexicográfica é a ordem em que palavras aparecem no dicionário. Ou seja, palavras idênticas até certa parte, como "programa" e "programação" são ordenadas de modo que a palavra de menor comprimento seja considerada "menor" e por isso venha antes na ordem.

Para buscar uma string em outra maior há a função StrMatch, que diferencia maiúsculas e minúsculas e a TextMatch, que não diferencia. Ambas aceitam coringas "?" e "*", onde a interrogação tem tamanho fixo e significa um caractere qualquer, duas interrogações, dois caracteres quaisquer. E asterísco significa qualquer caractere em qualquer quantidade.

A função FileMatch funciona da mesma forma que TextMatch, mas ignora o ponto "." Para trabalhar com nomes de arquivo no formato arquivo.extensão.

As funções StrMatchEx e TextMatchEx funcionam da mesma maneira que suas versões não estendidas, mas aceitam um tipo de sintaxe especial muito parecida com os ranges em Delphi. Por exemplo, 5:[a..z] para especificar cinco caracteres minúsculos ou 4:[0..9] para especificar quatro algarismos. A **Listagem 1** mostra todas as funções de comparação entre strings.

Não se limitando somente a comparação de strings também existem funções para busca de strings. Funções para busca de strings são especialmente úteis em bases de conhecimento e conversão de dados.

Para obter a posição de uma string dentro de outra existem duas opções, PosStr, case sensitive, e PosText, case insensitive. A função PosPchar tem o mesmo objetivo da PosStr/Text, mas trabalha com pchars. Ela que será mais usada na hora de trabalhar com arquivos na memória, trabalhar com a API do Windows ou trabalhar com uma quantidade grande de texto vinda de um arquivo, aberto através de uma stream.

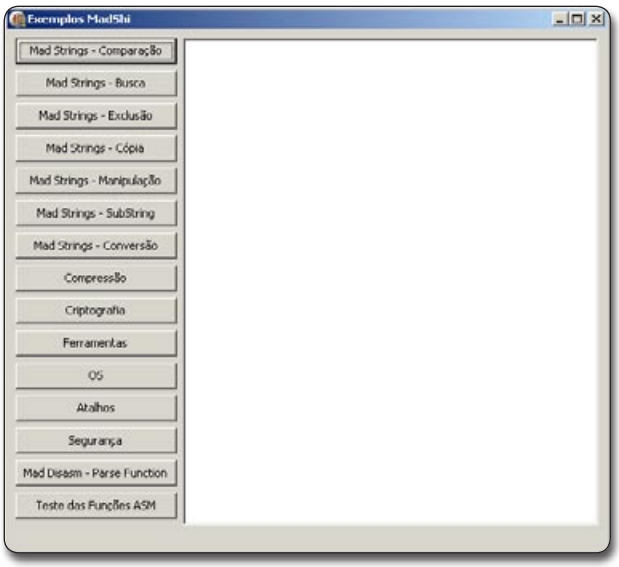


Figura 1. Formulário usado para testes das bibliotecas

Nota

Devido a alguns bugs na madCollection atual todo esse artigo foi feito baseado na madCollection Beta, que será a próxima a ser lançada. Os links para download de ambas as versões encontram-se na seção de links deste artigo. Depois de instalado o pacote madCollection instalará na pasta C:\Arquivos de programas\madCollection\madBasic\Help um help formado por um conjunto de arquivos HTML com explicações para cada biblioteca. O conteúdo desse help pode ser mais atual ou menos atual que o help encontrado no site dependendo da versão da biblioteca usada. Na seção de links pode ser visto o help online além de um fórum de discussão sobre a madCollection.

Nota

Para usar as funções CompareText e CompareStr da madShi e ainda assim usar a unit SysUtils deve-se chamar madShi.NomeDaFunção e SysUtils.NomeDaFunção. Como as duas units têm funções com o mesmo nome e assinatura uma chamada direta não compilaria por ambiguidade de sobrecargas, dando o erro: Ambiguous overloaded call to 'CompareStr'

Listagem 1. Alguns exemplos de funções de comparação de strings

```
if IsTextEqual('teste', 'TeStE') then
  mStrings.Lines.Add('IsTextEqual retorna que teste e TeStE são iguais')
else
  mStrings.Lines.Add('IsTextEqual retorna que teste e TeStE são diferentes');

mStrings.Lines.Add('CompareStr entre teste e TeStE '+inttostr( madStrings.
CompareStr('teste', 'TeStE')));
mStrings.Lines.Add('CompareText entre teste e TeStE '+inttostr( madStrings.
CompareText('teste', 'TeStE')));

str1 := 'programa';
str2 := 'PROGRAMA';
cmp := madStrings.CompareStr(str1, str2);
if cmp < 0 then
  mStrings.Lines.Add('A string ' + str1 + ' é menor que a ' + str2)
else if cmp =  0 then
  mStrings.Lines.Add('As strings ' + str1 + ' e ' + str2 + ' são iguais ')
else
  mStrings.Lines.Add('A string ' + str1 + ' é maior que a ' + str2);

str1 := 'programa';
str2 := 'programação';
cmp := madStrings.CompareText(str1, str2);
if cmp < 0 then
  mStrings.Lines.Add('A string ' + str1 + ' é menor que a ' + str2)
else if cmp =  0 then
  mStrings.Lines.Add('As strings ' + str1 + ' e ' + str2 + ' são iguais ')
else
  mStrings.Lines.Add('A string ' + str1 + ' é maior que a ' + str2);

str1 := 'programa';
str2 := 'PROGRAMA';
cmp := madStrings.CompareText(str1, str2);
if cmp < 0 then
  mStrings.Lines.Add('A string ' + str1 + ' é menor que a ' + str2)
else if cmp =  0 then
  mStrings.Lines.Add('As strings ' + str1 + ' e ' + str2 + ' são iguais ')
else
  mStrings.Lines.Add('A string ' + str1 + ' é maior que a ' + str2);

//validação de telefone com máscara
if StrMatch('(11)1234-5678', '(??)????-????') then
  mStrings.Lines.Add('As strings correspondem')
else
  mStrings.Lines.Add('As strings não correspondem');

//substring dentro de uma string, prefixos e posfixos
if TextMatch('programação', 'Programa???') then
  mStrings.Lines.Add('Os Textos correspondem')
else
  mStrings.Lines.Add('Os Textos não correspondem');

//arquivos com data embutida no nome
if FileMatch('foto-2010-01-01.gif', 'foto-????-??-??.gif') then
  mStrings.Lines.Add('Os arquivos correspondem')
else
  mStrings.Lines.Add('Os arquivos não correspondem');

//cpf com máscara
if StrMatchEx('612.385.888-24', '[3:0..9].[3:0..9].[3:0..9]-[2:0..9]')   then
  mStrings.Lines.Add('As strings correspondem')
else
  mStrings.Lines.Add('As strings não correspondem');

//cnpj com máscara
if TextMatchEx('08.997.921/0001-82', '[2:0..9].[3:0..9].[3:0..9]/[4:0..9]-[2:0..9]') then
  mStrings.Lines.Add('Os Textos correspondem')
else
  mStrings.Lines.Add('Os Textos não correspondem');
```

Listagem 2. Alguns exemplos de funções de busca de strings

```
var
  str1, str2: string;
  pos: integer;
begin

  str1:= 'Delphi';
  str2:=
    'A revista ClubeDelphi contém artigos sobre Delphi Win32 e ' +
    'Delphi Prism.';

  mStrings.Lines.Add('Se a posição for -1 significa que a string ' +
    'não foi encontrada.');
  mStrings.Lines.Add('a posição de ' + str1 + ' em ' + str2 + ' é ' +
    IntToStr(PosStr(str1, str2)));

  mStrings.Lines.Add('Posições em que aparecem o texto ' + str1 +
    'na string ' + str2);

  pos := 0;
  repeat
    pos :=  PosText(str1, str2, pos+1);
    if pos > 0 then
      mStrings.Lines.Add('Na posição ' + IntToStr(pos));
  until pos = 0;

  pos := PosPChar('teste', 'DUnit é usado em Testes unitários.', 0, 0, true);
  mStrings.Lines.Add('Posição de "teste" na frase: ' + IntToStr(pos));
```

Esta função aceita como parâmetros, além da substring a ser procurada e da string onde se procurará, os tamanhos de cada uma, onde começará a busca e onde terminará. Se os tamanhos das duas strings forem zero então a PosPchar internamente chamará StrLen para saber o tamanho de cada uma. Um outro parâmetro que ela aceita é um boolean para dizer se é case sensitive ou não. Seu uso é visto na **Listagem 2**.

O código, além de apresentar as funções de busca de strings, mostra como buscar por várias ocorrências de uma string dentro de um texto bem maior até que todas as ocorrências se esgotem. A função Delete do Delphi é muitas vezes usada sem que se perceba que é uma função nativa do IDE na unit System. A unit System é uma unit que sempre vem embutida em qualquer programa e não precisa ser declarada no uses. Isso dá a impressão que Delete é um comando que faz parte da linguagem, um comando direto do compilador, mas na verdade ela se comporta como procedure.

Delete aceita um parâmetro e o modifica. Na mad Collection há funções chamadas RetDelete, DeleteR e RetDeleteR. As começadas com "Ret" retornam a string nova, em vez de alterar o parâmetro de entrada. As terminadas em "R" deletam os caracteres a partir da direita. Há também a função KillChar para retirar da string todas as ocorrências de um determinado char e a função KillStr

Listagem 3. Exclusão de partes de strings

```
var
 str1: string;
 str2: string;
begin

 //deletar da string str1 5 caracteres a partir do  4°
 str1 := 'exemplo de uma string grande que deve ter um pedaço deletado';

 //função nativa do delphi
 mStrings.Lines.Add(str1);
 Delete(str1, 4, 5);
 mStrings.Lines.Add(str1);

 //função RetDelete
 mStrings.Lines.Add(RetDelete(str1, 1, 3));
 mStrings.Lines.Add(str1);

 //função RetDeleteR
 mStrings.Lines.Add(RetDeleteR(str1, 3));
 mStrings.Lines.Add(str1);

 //procedure DeleteR deleta a partir do final
 DeleteR(str1, 3);
 mStrings.Lines.Add(str1);

 str2 :='   exemplo de uma string com excesso de espaços   ';
 mStrings.Lines.Add(str2);
 mStrings.Lines.Add(RetTrimStr(str2));
 TrimStr(str2);
 mStrings.Lines.Add(str2);

 //se pelo menos um caractere de killchar for deletado o retorno é true
 //este é um exemplo em que todos os espaços são eliminados, inclusive no meio
 KillChar(str2, #32);
 mStrings.Lines.Add(str2);

 //retirando as palavras string da str1 e str2
 KillStr(str1, 'string');
 KillStr(str2, 'string');
 mStrings.Lines.Add(str1);
 mStrings.Lines.Add(str2);
```

para retirar todas as ocorrências de uma substring. Os exemplos podem ser vistos na **Listagem 3**.

As funções FillStr, ReplaceStr e ReplaceText podem ser muito úteis no dia-a-dia principalmente em relatórios, listagens, geração de páginas HTML ou exibição em tela de informação para que o usuário consulte. A **Listagem 4** mostra sua utilização.

Também existem as funções SubStr, SubStrCount e SubStrExists que trabalham contando ou obtendo substrings de uma string separada por um delimitador. O delimitador padrão, caso omitido, é "|". Outra parte interessante é uma coleção de funções para conversão de ou para inteiro e de ou para hexadecimal. O uso dessas funções é bem trivial e encontra-se no help, por isso serão omitidas neste artigo.

É importante deixar claro que as funções de conversão, apenas, da biblioteca madStrings trabalham com AnsiStrings em vez de UnicodeStrings. Warnings de conversão implícita nesse caso são sempre disparados, mas isso não é motivo de preocupação visto que em um número, decimal ou hexadecimal, os caracteres utilizados sempre são os algarismos de '0' a '9', o sinal de negativo '-', o separador de decimal, '.' Ou ',' e eventualmente as letras de 'A' a 'F'. Todos esses caracteres se encaixam na tabela ascii normal, de caracteres de um byte.

Compressão

As funções de compressão são triviais. Compress comprime uma string, Uncompress descomprime e Zip comprime um arquivo. É importante trabalhar com strings comprimidas apenas na memória, e se for gravar em arquivos, que se grave usando um stream e não um arquivo ini, arquivo txt ou TStringList. Isso porque a string "comprimida" não é uma string normal, mas sim uma cadeia de bytes que representam a string comprimida. Se essa cadeia contiver um caractere #0 no meio isso marcará, para controles, widgets, arquivos e editores, o fim da string e o restante será desprezado. Neste exemplo a string "É impossível para um homem aprender aquilo que ele acha que já sabe - Epíteto" foi comprimida para a cadeia de caracteres não imprimível "M#0#0#0#0#0#0# (...)" porém em um objeto TMemo aparecerá apenas "M".

No debugger do Delphi, copiando o valor da string pela janela de Local Variables ou Watches, os caracteres especiais de controle e não imprimíveis são representados por #$<Numero Hexadecimal>.

É muito importante comprimir apenas dados muito grandes. Isso porque dados de tamanho desprezível costumam ficar maiores quando comprimidos, devido à própria natureza dos algoritmos e estruturas de dados de compressão. No exemplo feito nesse artigo o tamanho comprimido fica maior que a string original. Usando uma string muito grande, ou uma string carregada de um arquivo de texto, é possível perceber os benefícios de se usar a compressão de strings. O exemplo de uso é visto na **Listagem 5**.

Listagem 4. Manipulação de strings

```
var
 str: string;
begin

 //fillStr preenche uma string, por exemplo com
   zeros
 mStrings.Lines.Add(FillStr('10,00', 8, '0'));

 //ReplaceStr e ReplaceText trocam uma string
   por outra
 str:= 'Delphi é um ambiente RAD e usa como
 linguagem de programação o Object ' +
  'Pascal, também conhecido como Delphi Language.';

 ReplaceText(str, 'Delphi', 'Visual Basic');
 mStrings.Lines.Add(str);
```

No exemplo são usadas três variáveis para fins didáticos: as três podem ser verificadas separadamente inclusive durante o debug. Mais esclarecedor do que usar apenas uma. No arquivo disponível para download é possível verificar um exemplo com o primeiro capítulo da obra "Memórias Póstumas de Brás Cubas" onde o tamanho da string comprimida caiu aproximadamente pela metade.

A função Zip aceita como parâmetros o caminho de um arquivo zip, que ela criará, um vetor contendo a lista de caminhos dos arquivos que serão comprimidos e o tercei-

ro parâmetro, opcional, é a lista de aliases que os arquivos terão. É útil para compactar um arquivo com um nome diferente do original. Poucos compressores fornecem essa opção tão facilmente.

Uma limitação do Zip do madCollection é que ele considera todos os arquivos no root, ele não salva dentro do arquivo "zipado" os caminhos relativos ou absolutos. Veja na **Listagem 6** seu uso. A diferença entre o segundo e o terceiro parâmetros é que o segundo é um array nativo, mas o terceiro é um tipo definido na unit madTypes chamado TDAUnicodeString que é um array dinâmico de UnicodeStrings. Outra diferença é o fato de os aliases não precisarem de caminhos completos.

Criptografia

A função Encrypt é usada para criptografar uma string, enquanto que Decrypt descriptografa. O resultado criptografado pode ter caracteres não legíveis, como os caracteres de controle e os brancos. Por isso não pode ser exibido em um TMemo ou gravado em um arquivo, pode apenas ser comparado ou usado na memória, em variáveis.

Para transformar essa string (ou buffer) binário em uma string válida e legível a função Encode transforma essa string em uma string base 64, onde cada caractere será codificado de maneira especial usando-se caracteres válidos, da mesma forma como são codificados os anexos de um e-mail. Isso fará com que o tamanho do texto criptografado aumente. A função Decode transforma o texto base 64 em uma string binária para que possa ser novamente descriptografada, acompanhe na **Listagem 7**. As funções Encrypt, Decrypt, Encode e Decode trabalham apenas com AnsiStrings.

Tools

Quatro funções especialmente úteis são: GetFileVersion, FileVersionToStr, GetLongFileName e GetShortFileName. GetFileVersion retorna a versão de um arquivo informado, FileVersionToStr pode transformar essa informação em uma string, GetLongFileName pode trazer o nome "longo" de um arquivo através de seu nome curto e GetShortFileName faz o contrário. Isso é ideal para interoperabilidade com sistemas legados em DOS, ou para gerar arquivos em lote (*.bat) que executem uma sequência de programas legados passando arquivos como parâmetro. Seu uso está na **Listagem 8**.

Esta não é a única funcionalidade provida pela unit madTools. Há também duas funções interessantes que convertem um método para procedure e procedure para método. Para esclarecer a diferença entre métodos e procedures (ou functions) estáticos e globais convém pensar que, em código de máquina, mesmo um método e uma procedure idênticos têm assinaturas diferentes.

Considerando a function soma(a,b: integer): integer e o método da classe TFoo, TFoo.soma(a,b: integer):integer. Embora o método

Listagem 5. Compressão de strings

```
var
  str: string;
  strComp: string;
  strDescomp: string;
begin
  str := 'É impossível para um homem aprender aquilo que ele acha que
  já sabe - Epíteto';
  strComp := Compress(str);
  strDescomp := Uncompress(strComp);

  mStrings.Lines.Add('String original: ' +  str);
  mStrings.Lines.Add('Comprimento: ' + IntToStr(Length(str)));

  mStrings.Lines.Add('String Comprimida: ' +  strComp);
  mStrings.Lines.Add('Comprimento: ' + IntToStr(Length(strComp)));

  mStrings.Lines.Add('De volta à string original: ' + strDescomp);
  mStrings.Lines.Add('Comprimento: ' + IntToStr(Length(strDescomp)));
```

Listagem 6. Compressão de arquivos

```
var
  arquivoZip: string;
  arquivos: array[1..1] of string;
  aliases: TDAUnicodeString;
begin

  //nome do próprio programa, como exemplo o próprio executável será comprimido
  arquivos[1] := ParamStr(0);

  //arquivo zip
  arquivoZip := ExtractFilePath(ParamStr(0))+'\arquivo.zip';
  //zip sem alias
  Zip(
     arquivoZip,
     arquivos,
     nil
   );

  //arquivo zip
  arquivoZip := ExtractFilePath(ParamStr(0))+'\arquivocomalias.zip';

  SetLength(aliases, 1);
  aliases[Low(aliases)] := 'alias.exe';

  //zip com alias
  Zip(
     arquivoZip,
     arquivos,
     aliases
   );
```

Listagem 7. Criptografia

```
var
  AlgoSigiloso: AnsiString;
  Senha: AnsiString;
  AlgoBase64: AnsiString;
begin

  AlgoSigiloso := 'Você guarda um segredo? Eu também!';
  Senha := 'Delphi'; //não use senhas óbvias

  mStrings.Lines.Add('Texto sigiloso: ' + AlgoSigiloso);
  Encrypt(AlgoSigiloso, senha);
  mStrings.Lines.Add('Texto criptografado (não legível): ' + AlgoSigiloso);
  AlgoBase64 := Encode(AlgoSigiloso);
  mStrings.Lines.Add('Texto criptografado em base 64: ' + AlgoBase64);

  AlgoBase64 := Decode(AlgoBase64);
  Decrypt(AlgoBase64,Senha);
  mStrings.Lines.Add('Texto Descriptografado: ' + AlgoBase64);
```

Listagem 8. Arquivos

```
  mStrings.Lines.Add('Versão de calculadora: ' + FileVersionToStr(GetFileVersion('
c:\windows\system32\calc.exe')));
  mStrings.Lines.Add(GetShortFileName(paramstr(0)));
  mStrings.Lines.Add(GetLongFileName(GetShortFileName(paramstr(0))));
```

e a função estática sejam iguais quanto à sua forma e assinatura, elas são diferentes binariamente. A função que faz parte de uma classe está em uma área diferente da memória, e tem um parâmetro a mais implícito, que não precisa ser passado, que é o "self", uma referência à instância do objeto à qual ela mesma faz parte. Em alguns casos é necessário atribuir uma procedure a um evento de mesma assinatura.

No help da madCollection é mostrado o exemplo mais clássico: atribuir um mesmo comportamento, através de uma procedure, ao tratamento de exceções de todo o programa.

Para isso se deveria atribuir ao Application.OnException alguma procedure, mas essa não poderia ser uma procedure estática, deveria ser ou membro dinâmico de uma classe que serviria somente para isso, impondo ao programador o overhead de instanciar um objeto dessa classe. Ou então fazer um método de classe, com a única vantagem de não ser mais necessário criar uma instância, mas sem se livrar da tarefa de escrever.

Com o método ProcedureToMethod é possível "transformar" uma procedure estática em um método de objeto, para que possa ser atribuído a um evento. Assim, o evento que vai tratar as exceções de todo o projeto pode ser definido em uma procedure estática.

Isso é útil quando o evento em si não faz parte da regra de negócio de nenhuma classe e pode ser colocado em uma procedure estática global em uma unit genérica. Um exemplo seria duas procedures para fazer qualquer Edit de todos os formulários ficarem amarelos quando se entra neles (OnEnter) e voltarem a ficar brancos quando se sai deles (OnExit) destacando para o usuário qual é o Edit em foco.

Há também uma function que faz o caminho inverso: transforma um método em uma procedure, suprimindo o parâmetro que faria referência ao próprio objeto. Isso é útil no uso de APIs do Windows que aceitam uma referência ou ponteiro para uma função.

O exemplo da **Listagem 9** mostra como o método Add de um TMemo pode ser transformado em procedure, como se fosse uma "ShowMessage". O código usa o método AddMensagem, bem trivial, que simplesmente adiciona no memo da form a mensagem desejada. Depois é declarada a variável ShowMessageNoMemo como uma procedure (const s: string). Essa procedure recebe o endereço e o parâmetro implícito self de AddMensagem com o uso de MethodToProcedure. Isso transforma ShowMessageNoMemo numa procedure estática global igual a ShowMessage. Isso é apenas um exemplo. A variável ShowMessageNoMemo não é visível pelas outras units, e além disso precisaria ser destruída com VirtualFree(ShowMessageNoMemo, 0, MEM_RELEASE) na seção finalization da unit. Ao contrário da ProcedureToMethod, a MethodToProcedure só funciona se tanto o método como a procedure forem stdcall.

Listagem 9. Transformando um método em procedure

```
procedure TfrmExemplos.AddMensagem(const s: string);  stdcall;
begin
  mStrings.Lines.Add(s);
end;

var
  ShowMessageNoMemo : procedure(const s: string); stdcall;
  p: pointer;
begin

  p := @TfrmExemplos.AddMensagem;
  ShowMessageNoMemo := MethodToProcedure(self, p);

  ShowMessageNoMemo('Versão de calculadora: ' + FileVersionToStr
  (GetFileVersion('c:\windows\system32\calc.exe')));
  ShowMessageNoMemo(GetShortFileName(paramstr(0)));
  ShowMessageNoMemo(GetLongFileName(GetShortFileName(paramstr(0))));
```

OS

Dentro da Unit madTools existe uma função chamada OS que devolve um Record do tipo TOS. TOS tem onze membros:

- Major – é do tipo cardinal e traz a versão "maior" ou o número mais relevante da versão do SO;
- Minor – é cardinal também e traz o número menos significativo da versão (o segundo);
- build – é o número de build do sistema, e o menos significativo;
- spStr – string, é a descrição textual do service Pack;
- win9x - Boolean, true se o Windows for do tipo / série Win 9X (95, 98, ME);
- win9xEnum – Enum do tipo TOsEnum que contempla todos os tipos do Windows, exceto os mobile;
- winNt - True se for do tipo winNT, como Windows NT, 2000, 2003, 2008, XP, Vista e 7;
- winNtEnum – analogamente a win9XEnum, TOsEnum;
- Enum – igual a win9xEnum e winNt Enum;
- x64 – Boolean, true se for versão de 64 bits;
- spNo – Número do service pack;
- description – Descrição textual da versão instalada do Windows.

Esta função é especialmente útil na hora de verificar a versão do sistema operacional para a execução ou não de alguma rotina que dependa estritamente de alguma versão específica do sistema operacional. Confira na **Listagem 10**.

Esta não é a única unit para se trabalhar com recursos do sistema operacional.

Nota

As strings no Delphi podem ter até MAXINT - 8 caracteres, ou seja, 4294967287. Porém strings com "apenas" 25% desse tamanho já pesam mais de um gigabyte na memória são por si um suplício para qualquer computador. Há maneiras mais elegantes e performáticas de lidar com strings enormes usando streams e buffers.

Nota

Eventos de uma classe ou componente nada mais são do que propriedades. Essas propriedades são de um tipo de dado especial (procedure of object) que são na verdade referências a métodos.

Nota

Uma soma ou outra expressão matemática qualquer é muito mais trivial do que um "Hello World" em qualquer linguagem de programação, em qualquer computador. Isto se deve ao fato de que imprimir "Hello World" em uma janela, tela ou pop-up trabalha com strings, e strings sempre são estruturas de dados complexas, que podem ser desde vetores de tamanho variável terminados em #0 até as complexas UnicodeStrings do Delphi.

Outros recursos estão espalhados pelas units madSecurity, madShell, madRemote e madKernel.

Shell

A unit madShell oferece muitas facilidades. A função GetSpecialFolder permite obter o caminho de qualquer diretório especial do Windows, mesmo que durante a instalação tenha havido modificações dos caminhos padrão ou que o usuário tenha mudado isso no registro.

A função NewShortCut cria um atalho. Podem ser definidos o diretório de trabalho, estado da janela principal, ícone, tecla de atalho entre outras informações. Vários programas, como o µTorrent, criam um ícone na área de trabalho depois da primeira execução. O exemplo da **Listagem 11** mostra como criar um ícone para o próprio programa em execução que abra maximizado e que tenha como tecla de atalho ctrl+[. É uma tecla de atalho bem sugestiva: ctrl+ "abre colchete" abre a janela do programa.

Listagem 10. Função OS

```
var
  meuOs: TOS;
begin

  meuOs := OS;
  mStrings.Lines.Add('Tipo/Nome: ' + GetEnumName(TypeInfo(TOsEnum),
Ord(meuOs.Enum)));
  mStrings.Lines.Add('Versão: ' + IntToStr(meuOs.major)+"."+IntToStr(meuOs.
minor) + "." + IntToStr(meuOs.build));
  mStrings.Lines.Add('SP: ' + meuOs.spStr);
  mStrings.Lines.Add('SPno: ' + IntToStr(meuOs.spNo));
  mStrings.Lines.Add('Descrição: ' + meuOs.description);

  if meuOs.win9x then
  begin
    mStrings.Lines.Add('Serie Win 9x');
    mStrings.Lines.Add('Tipo/Nome: ' + GetEnumName(TypeInfo(TOsEnum),
Ord(meuOs.win9xEnum)));
  end
  else
  if meuOs.winNt then
  begin
    mStrings.Lines.Add('Serie Win NT');
    mStrings.Lines.Add('Tipo/Nome: ' + GetEnumName(TypeInfo(TOsEnum),
Ord(meuOs.winNtEnum)));
  end;
```

Listagem 11. Criando atalho

```
var
  AreaDeTrabalho: AnsiString;
begin
  GetSpecialFolder(sfDesktopDir, AreaDeTrabalho);
  with NewShortCut(ParamStr(0)) do
  begin
    WorkingDir := ExtractFilePath(ParamStr(0));
    ShowCmd    := SW_SHOWMAXIMIZED;
    HotKey := 'ctrl+[';
    Save(AreaDeTrabalho+'\programa.lnk', true);
  end;
```

Listagem 12. Obtendo o usuário corrente

```
  mStrings.Lines.Add('Usuário Corrente:');
  mStrings.Lines.Add('Nome: ' + CurrentUser.Name);
  mStrings.Lines.Add('Sid: ' + CurrentUser.SidStr);
  mStrings.Lines.Add('Tipo: ' +  GetEnumName(TypeInfo(TAccountType),
  ord(CurrentUser.Type_)));
```

Segurança

A unit madSecurity fornece tudo o que é necessário para se trabalhar com permissões de arquivos, serviços, chaves de registro, compartilhamentos, impressoras e qualquer outro objeto que necessite de autorização. Também é possível obter o usuário corrente e listar usuários logados. Outra utilidade seria listar compartilhamentos de um servidor, da própria máquina ou criar um compartilhamento de pasta ou de impressora. A **Listagem 12** apenas lista o usuário corrente, colocando seu nome e ID no memo.

Disasm

Para terminar este artigo com chave de ouro é necessário falar sobre a unit madDisasm. As duas funções que interessam dessa unit são a ParseCode e a ParseFunction. Enquanto ParseCode traz uma instrução assembly que esteja imediatamente no endereço especificado, ParseFunction traz toda uma procedure ou function a partir deste endereço.

A função ParseCode retorna uma estrutura do tipo TCodeInfo, e a ParseFunction retorna uma estrutura do tipo TFunctionInfo. Essas estruturas representam o funcionamento interno de um programa dentro do processador e tem informações como: onde o código começa, qual o seu tamanho, se é válido e assim por diante.

Essas estruturas de retorno não são consideradas neste artigo, pois muito conhecimento teórico de assembly e da estrutura x86 seriam necessários. Como já dito antes, a unit madDisasm foi criada para ser usada pela madCodeHook e pela madExcept, e só serve para hacks avançados.

Um dos parâmetros passados tanto para a ParseCode como para a ParseFunction é uma string, passada por referência. Ela precisa ser uma AnsiString, que é a string "antiga" do Delphi. Nessa string são armazenados os opcodes e instruções da função em assembly. Considerando a função soma, estática e global:

```
function soma(a, b: integer): integer;
begin
  Result := a+b;
end;
```

Aplicando-se ParseCode e ParseFunction nessa função teremos o resultado da **Listagem 13**.

ParseCode traz apenas a primeira instrução, que adiciona o valor no registrador EAX ao valor do registrador EDX, mas não move o resultado para o Result da função e nem "sai" da função. ParseCode traz realmente a primeira instrução que roda nesse ponteiro. Parse function traz a função completa. É possível remover o entrypoint da função e os offsets e criar uma função em assembly no Delphi assim:

```
function somaASM(a,b: integer): integer;
asm
  add    edx, eax
  mov    eax, edx
  ret
end;
```

Listagem 13. Aplicando ParseCode e ParseFunction

```
type
 //Tipo que define a função soma
 Func = function (a,b: integer): integer;
 pFunc = ^Func;
var
 sAsmCodeSoma: AnsiString;
 codeSoma: Func;
 pSoma: pFunc;
begin
 //obtendo um ponteiro para a função soma e decodificando-a
 codeSoma := soma;
 pSoma := @codeSoma;

 mStrings.Lines.Add('********** ParseCode ********');
 ParseCode(pointer(pSoma), sAsmCodeSoma);
 mStrings.Lines.Add(sAsmCodeSoma);

 mStrings.Lines.Add('');
 mStrings.Lines.Add('********** SEPARADOR ********');
 mStrings.Lines.Add('');

 mStrings.Lines.Add('********** ParseFunction ********');
 ParseFunction(pointer(pSoma), sAsmCodeSoma);
 mStrings.Lines.Add(sAsmCodeSoma);


********** ParseCode ********
004b4a80  add    edx, eax


********** ParseFunction ********
004b4a80 sub_4b4a80:              ; function entry point
004b4a80  add    edx, eax
004b4a82  mov    eax, edx
004b4a84  ret
```

Nota do DevMan

EAX, EBX, ECX e EDX são os nomes de alguns dos registradores do processador. A grosso modo pode-se dizer que são "variáveis de fábrica". No Delphi o retorno das funções (ou um ponteiro para tal) fica no registrador EAX. EAX também é o primeiro parâmetro de uma função, porém quando existem mais parâmetros do que registradores, depois que o último parâmetro é alocado em um registrador, os demais são empilhados na memória.

Essa função tem o mesmo efeito: somar dois valores. Se for depurada com as opções step into, na primeira linha será adicionado o valor 48 (em eax) ao registrador edx (que já tem 13) ficando com o valor de 61. Na segunda linha o valor 61 de edx é movido para eax e na terceira linha a função retorna. O Result é o que está em eax.

Já é possível perceber o poder e flexibilidade que o módulo madDisasm confere. Uma rápida olhada na internet revela que alguns vírus foram feitos usando a biblioteca madShi, e é por isso que foram impostas restrições mais rígidas quanto à sua distribuição.

Conclusão

Existem muitas outras funções na biblioteca, mas duas que merecem destaque são EnumProcess da uni madRemote e RegisterShellEvent da unit madShell. A primeira lista todos os processos rodando, e a segunda faz um log de todos os arquivos que estão sendo criados, renomeados, deletados ou alterados. Os exemplos podem ser conferidos nos arquivos disponíveis para download.

Alguns métodos da madShi estão bem documentados, mas carecem de exemplos. O autor da biblioteca pressupõe que quem usa a madShi tem esses conhecimentos avançados, sendo capaz de criar um antivírus ou anti-spyware. Sendo comprador da versão paga é possível obter um suporte mais personalizado, com certeza.

A maioria dos métodos ficou de fora dos exemplos desse artigo por exigirem um código muito verboso com algumas linhas simplesmente "esotéricas", além de exigirem longas explicações teóricas.

Com esses conhecimentos será possível acelerar o desenvolvimento de certos módulos, aproveitando-se as funções de strings, por exemplo, já prontas, e será possível usar facilmente recursos avançados e pouco documentados do Windows que podem facilitar tarefas cotidianas e ainda impressionar clientes e outros programadores.

Se alguém perguntar "Como é se que faz isso?" Simplesmente responda que é mágica.

Vitor Luiz Rubio
vitorrubio@gmail.com
http://blog.vitorrubio.com.br
http://www.vitorrubio.blogspot.com/
http://twitter.com/vitorrubio

Tecnólogo em Processamento de Dados pela FATEC-SP e MCP. Atua como Analista de Sistemas Sênior na Editora Revista dos Tribunais. Programa com Delphi desde a versão 3 e com .Net desde a versão 1.1.

madShi
http://madshi.net/

Help madShi
http://help.madshi.net/
http://help.madshi.net/Index.htm

Mad Collection Beta
http://madshi.net/madCollectionBeta.exe

Mad Collection Atual
http://madshi.net/madCollection.exe

Forum
http://forum.madshi.net/

News
news://news.madshi.net

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Padrões Criacionais
Simplicidade com Padrão Buil

Talvez um dos livros mais importantes (pelo menos o mais famoso), sobre os padrões de projetos seja "Design patterns: elements of reusable object-oriented software" (em português, "Padrões de Projeto: Elementos de Software Orientado a Objetos Reutilizável"), publicado em 1994. Nele os autores Erich Gamma, Richard Helm, Ralph E. Johnson e John Vlissides descrevem 23 padrões de projetos divididos em três categorias: padrões criacionais, padrões estruturais e padrões comportamentais.

Estes padrões normalmente trabalham juntos, pois dificilmente você terá um problema cuja solução seja simplesmente implementar um determinado padrão. Na maioria das vezes, você precisa compreender o conceito apresentado pelo padrão. Só assim você será capaz de interpretá-lo e identificar situações para usá-lo no seu projeto.

Os 23 padrões publicados neste livro receberam o apelido de padrões GoF (Gang of Four – Gangue dos Quatros), devido aos quatro autores do livro. "Design patterns: elements of reusable object-oriented software" difundiu o conceito de padrões de projeto, que até aquela época estava sob o domínio de poucos. Por conta disto, a publicação do livro é considerada por muitos como um divisor de águas.

Os padrões GoF são os padrões mais populares, mas existem outros como, por exemplo os padrões GRASP (General Responsibility Assignment Software Patterns - Padrões de Princípios Gerais para Atribuição de Responsabilidade). Dentre os vários padrões GRASP, os mais populares são: Creator, Information Expert (ou apenas Expert) e Controller. Enquanto os padrões GoF exploram soluções bem específicas, os padrões GRASP exploram as práticas e princípios da orientação a objetos.

Resumo DevMan

De que trata o artigo:

O artigo trata da implementação do design pattern Builder, um padrão criacional que se tornou popular após o lançamento do livro "Padrões de Projeto: Elementos de Software Orientado a Objetos Reutilizável", da famosa "Gangue dos quatro" (ou GoF, em inglês).

Para que serve:

Todo projeto precisa criar instâncias de classes, mas se isto não for feito de forma controlada, o projeto tende a tornar-se complexo e rígido, dificultando a manutenção do mesmo. Os padrões criacionais são técnicas que ajudam controlar como as instâncias de classes são criadas.

Em que situação o tema é útil:

O Padrão Builder é útil quando você se deparar com um objeto complexo que precisa de muitas informações para ser criado. Neta situação o Padrão Builder é capaz de simplificar a criação deste objeto ao mesmo tempo em que adiciona mais flexibilidade ao projeto e controla como as instâncias de classes são criadas.

Builder:

Criar softwares seguindo os princípios de orientação a objetos bem como os padrões de projeto é meio caminho rumo ao software eficiente. Mas nem sempre é fácil identificar quando e onde aplicar um princípio ou padrão de projeto. A melhor maneira de identificar onde usar um ou outro dos diversos padrões existentes é, em primeiro lugar, conhecer o máximo de padrões possíveis. Neste artigo você aprenderá como implementar o Padrão Builder.

Do que é feito um padrão

Um padrão de projeto não passa de uma solução flexível e reutilizável para um problema de projeto. Digo "problema de projeto", por que os padrões de projetos não são soluções para o software em si, por exemplo, um padrão de projeto não diz como um software deve estabelecer conexão com o banco de dados, mas ele diz como o projeto deve ser codificado para que a camada de conexão seja flexível o máximo possível. Enfim, um padrão de projeto soluciona um problema de projeto.

Além de abordar e solucionar um problema de forma a tornar o projeto mais flexível, um padrão ainda precisa ter um nome e uma descrição das consequências – boas e/ou ruins – da sua implementação. Todo padrão de projeto terá no mínimo estas quatro características: nome, problema, solução e conseqüências:

- Nome: resume o problema, a solução e/ou a conseqüência do uso;
- Problema (ou motivação): descreve a situação de projeto em que o padrão pode ajudar se for implementado;
- Solução (ou intenção): mostra como aplicar os padrões descrevendo quais são os participantes (classes), do padrão e qual o papel de cada um;
- Consequências: descreve qual o impacto no projeto devido ao uso deste padrão – é normal que um padrão tenha consequências boas e também ruins.

der

Rafael Stavarengo

O nome diz muito a respeito de um padrão. Quando você lê "Padrão Observer", o que mais poderia pensar além de que este padrão faz uma classe observar outra classe. O que não é possível saber apenas olhando o nome do padrão é como ele faz o que diz fazer. Para isto serve a descrição do problema em que ele se aplica.

A descrição do problema é sempre a mais abstrata possível, para que qualquer programador de qualquer linguagem seja capaz de compreendê-la. Isto quer dizer que você não vai encontrar uma descrição de problema como esta: "Se você estiver precisando criar projeto com a capacidade de trocar de aparência, use o padrão Abstract Factory.". Não teria graça se fosse assim. Uma descrição mais real é: "Se precisar criar uma família de objetos sem depender de quem são os integrantes desta família ou de como eles são criados, use o padrão Abstract Factory.". A diferença é que a primeira frase descreve UM PROBLEMA, enquanto que a segunda frase descreve O PROBLEMA que o padrão Abstract Factory pode ajudar a resolver.

Depois de interpretar a descrição do problema e concluir que ele pode ajudar o seu projeto, você vai precisar conhecer quais são as consequências de usá-lo. Talvez você ache que as consequências são ruins ou que uma das consequências seria "cara demais" para o projeto – se chegar a esta conclusão o melhor a fazer é procurar por outro padrão, pois diversos padrões diferentes podem solucionar um mesmo problema, cada um com suas consequências – boas e ruins – de uso.

Por fim, resta a descrição da solução, cujo objetivo é explicar como implementar o padrão de projeto. A descrição da solução é tão abstrata quanto a descrição do problema, por isso é necessário interpretar a solução ao invés de decorá-la. A solução apresenta, por meio de um diagrama de classes, quais são os participantes (as classes), da solução além de uma explicação de qual é o papel de cada participante.

As categorias

Além das características que você leu no último tópico "Do que é feito um padrão", os padrões de projetos GoF também pertencem a uma destas três categorias: criacionais, estruturais e comportamentais:

- Criacionais: oferecem mecanismos diferentes para separar a criação de objetos;
- Estruturais: definem maneiras de criar composições complexas ou adicionar comportamentos a um objeto sem necessariamente precisar alterar o código da classe;
- Comportamentais: padrões especializados em comunicação entre os objetos – definem maneiras dos objetos se comunicarem sem precisar conhecer muito (ou nada), um do outro.

Os padrões comportamentais são voltados a comunicação entre os objetos e classes. Um projeto tende a entrar em colapso quando a comunicação e interação entre seus objetos e classes não está bem projetada. Os padrões GoF definem 11 padrões comportamentais, cujos mais populares são: Command, Observer, Strategy e Iterator.

Já com os padrões estruturais você consegue usar da herança para criar interfaces compostas. Isto significa ter a capacidade de usar a composição de objetos para criar novas funcionalidades em tempo de execução.

Os padrões criacionais, que inclui o Builder, são padrões com objetivo de diminuir a dependência do construtor das classes, isto quer dizer que, quando se implementa um padrão criacional o número de instruções como esta "`TMinhaClasse.Create`" tende a diminuir. Veja a seguir.

Nota

O problema relacionado ao uso do construtor das classes é apenas um dos problemas cujos padrões criacionais ajudam amenizar.

Padrões criacionais

Usar o construtor das classes sem controle é considerado uma prática ruim. O princípio de orientação orientada a objetos "Programe Para Interfaces" defende que, um projeto reutilizável deve ser baseado no uso das interfaces, e não das implementações, porém, toda vez que o construtor de uma classe é usado, obviamente uma implementação está sendo criada, já que apenas elas podem ser instanciadas por um construtor.

Por outro lado, como é possível usar uma classe sem invocar o seu construtor? Este é um problema de projeto pelo qual você não precisa passar, pois a solução para ele foi encontrada, testada e documentada. Hoje esta solução é usada por milhares de programadores em todo o mundo: são os padrões criacionais.

Objetos complexos

O objetivo do Padrão Builder é facilitar a criação de objetos complexos. Alguns objetos necessitam mais do que uma simples instrução `TMinhaClasse.Create` para funcionar corretamente. Na **Listagem 1** está um exemplo de objeto complexo e a **Listagem 2** mostra como o padrão Builder simplifica a construção dele.

Nota do DevMan

A **Listagem 2** mostra uma implementação do padrão Builder, conhecida como Query Object. Ou seja, existe um problema muito comum, tão comum que a solução para este problema ganhou um apelido: Query Object. Esta técnica tem como objetivo criar um critério de pesquisa usando objetos. Assim retira a necessidade do programador conhecer as particularidades da linguagem de pesquisa, como a linguagem SQL.

Listagem 1. Criando um objeto complexo

```
01  var SqlObject: TSqlObject;
02  begin
03    SqlObject = TSqlObject.Create
      ('*', 'clientes', 'idade > 21', 'nome desc');
04    SqlObject.ToString;
05  end;
```

Listagem 2. Padrão Builder em ação

```
01  var SqlObject: TSqlObject;
02  begin
03    SqlObject = TSqlObject.Create;
04    SqlObject.Columns('*');
05    SqlObject.From('clientes');
06    SqlObject.Where('idade > 21');
07    SqlObject.OderBy('nome desc');
08    //Build retorna uma string contendo
      a query SQL
09    SqlObject.Build;
10  end;
```

Na **Listagem 1**, para conseguir criar uma instância da classe TSqlObject, o programador precisa passar muitos parâmetros para o construtor da classe. Isso é necessário porque a classe TSqlObject precisa de muita informação para funcionar. O código com certeza funciona, mas é desastroso, pois o projeto fica preso a uma única implementação de TSqlObject, o que vai gerar enormes dificuldades quando o banco de dados for atualizado para uma versão com sintaxe SQL diferente. A própria aparência do código alerta que algo precisa ser melhorado.

Essa melhoria pode ser conseguida com o Padrão Builder – veja na **Listagem 2**. Apesar de o projeto continuar preso à implementação TSqlObject, ficou mais simples trocar o uso desta implementação pelo uso de uma interface. É possível ver uma grande melhoria, inclusive de aparência. Com o Padrão Builder ficou mais simples compreender o que está acontecendo no código apenas olhando para ele.

Padrão Builder

Vou direto ao assunto: apresento-lhe o Padrão Builder (**Figura 1**).

Para identificar locais onde implementar um padrão, seja lá qual for ele, você precisa ter compreendido-o – não apenas decorado. Isto é assim, porque a implementação de um padrão precisa ser adaptada à realidade do projeto, portanto é necessário que você tenha compreendido o conceito por trás do padrão para poder fazer as adaptações necessárias.

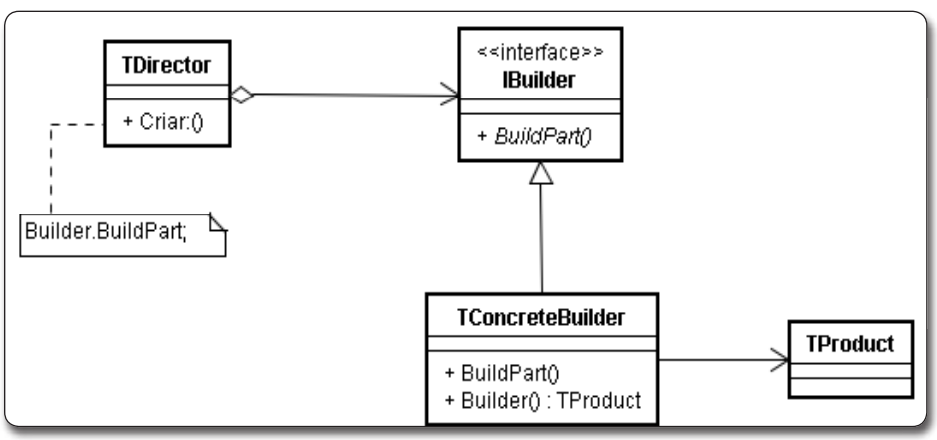


Figura 1. Design do Padrão Builder

A primeira adaptação é substituir os participantes do diagrama de classe pelas verdadeiras classes existentes no projeto. No caso do Padrão Builder, existem no mínimo quatro participantes: TDirector, TBuilder, TConcreteBuilder e TProduct.

O participante TDirector é uma classe também conhecida como "classe cliente". Ela representa uma classe do projeto que usará o Padrão Builder. Ela pode ser qualquer classe, formulário ou não formulário. Podem existir várias classes que usam o Builder: todas elas são o que TDirector está representando. O método `Criar` que aparece em TDirector, é apenas um exemplo de um método que usa o Builder para construir algo. Este método não precisa existir de verdade porque está no diagrama apenas como exemplo.

A interface IBuilder é muito importante, apesar de existirem maneiras de implementar o Padrão sem ela. Se ela não existirem o projeto estará preso às implementações e qualquer padrão pouco ajuda quando o projeto está nessa condição. Na verdade fica mais difícil implementar um padrão quando não se respeita o princípio programe para interfaces. Um projeto assim dificilmente será flexível e reutilizável. Mais adiante, você entenderá a importância de se respeitar o princípio programe para interfaces.

O participante TConcreteBuilder pode ser considerado o mais importante do Padrão, pois é ele quem faz o "trabalho pesado". TConcreteBuilder representa as classes que simplificam o processo de criar/construir um objeto complexo, ou seja, ele é o Builder (construtor), propriamente dito. Por fim, TProduct representa a classe que será criada por TConcreteBuilder.

Em resumo, tudo funciona assim: TDirector, pede a uma das implementações de IBuilder para fabricar uma instância da classe TProduct. A seguir, aprenda como implementar Padrão Builder vendo um exemplo de como tudo funciona na prática.

O exemplo

Para exemplificar como implementar o Padrão Builder e mostrar na prática quais são as conseqüências boas e ruins de sua

implementação, criei uma aplicação com problemas reais do cotidiano. A aplicação começa com poucas funções e à medida que as quantidades de requisitos aumentam, torna-se necessário implementar algum padrão comprovado para tornar o projeto mais flexível. Obviamente será usado o Padrão Builder.

O exemplo se trata de uma aplicação usada por uma lanchonete fictícia. Esta lanchonete vende os seguintes produtos: hambúrguer, batata frita, refrigerante ou combo, que são os três produtos juntos. O objetivo do sistema é cadastrar os pedidos dos clientes. Tudo muito simples, como deve ser. Veja o único formulário da aplicação, na **Figura 2**.

Figura 2. O formulário para lançar pedidos

O Sistema de Pedidos é muito simples de usar. O usuário escolhe um dos produtos disponíveis, informa quantidade e em seguida clica no botão 'Adicionar ao pedido'. Fazendo isto o produto escolhido é adicionado no grid. Quando o usuário escolher o produto 'Combo', três linhas serão adicionadas ao grid, pois o 'Combo' representa a venda do hambúrguer, da batata frita e do refrigerante juntos.

Veja o diagrama de classes do Sistema de Pedidos na **Figura 3**. Neste diagrama é possível descobrir muita informação sobre o Sistema de Pedidos. Para começar, os produtos são representados por uma interface criada pela classe abstrata TProdutoAbstract. Isto é bom, mas por outro lado o projeto não está usando este benefício, pois a classe TForm1 depende de todas as implementações de TProdutoAbstract. Isto revela que o projeto não está respeitando o princípio "Programe para Interfaces".

Como o sistema é exageradamente simples, esta falta de princípios parece não estar causando problemas. Será? A única maneira de colocar isto à prova é fazendo algumas alterações para sentir o grau de dificuldade ao executá-la. Mas, antes das alterações começarem, veja as listagens seguintes como o projeto está codificado e entenda porque a classe TForm1 depende das implementações de TProdutoAbstract.

A classe TForm1 possui apenas três métodos: FormCreate (**Listagem 3**), BotaoAdicionarClick (**Listagem 4**) e AdicionarProdutoAoGrid (**Listagem 5**). O grid usado no formulário é um componente TStringGrid, o qual usa poucos recursos de hardware ao mesmo tempo em que oferece poucas funções prontas ao programador. Quando o formulário é criado, o método FormCreate é invocado. Este método determina o nome e tamanho de cada coluna do grid. A quantidade de colunas que o grid terá já está definido na propriedade ColCount do grid.

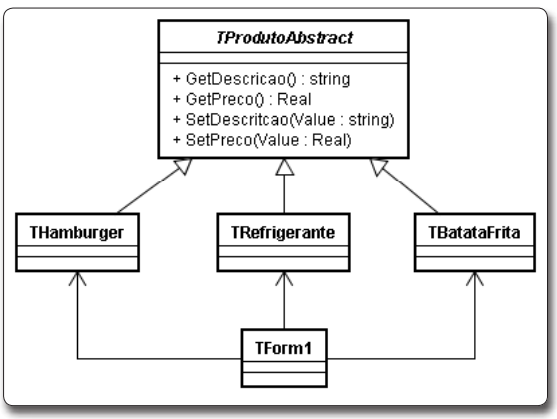


Figura 3. O desing do projeto

Depois que o sistema está em execução, o usuário pode lançar os pedidos. Este processo é executado em dois passos: primeiro o sistema determina qual implementação de TProdutoAbstract deve ser criada de acordo com o produto selecionado (**Listagem 4**) pelo usuário e depois as informações do produto escolhido são lançadas no grid (**Listagem 5**).

Repare entre as linhas 13 e 15 da **Listagem 4** que o produto "Combo" corresponde a uma instância de todos os produtos. Nesta listagem está o ponto exato onde o princípio "Programe para interfaces" é quebrado: o formulário (a classe TForm1), conhece todas as implementações de TProdutoAbstract existentes. Isto não está causando nenhum problema devido à simplicidade do projeto, mas projetos maiores e mais complexos poderiam ter sérios problemas.

Nas linhas 5, 7 e 9 da **Listagem 4**, o método AdicionarProdutoAoGrid é invocado. Este método pode ser visto na **Listagem 5**. Ele recebe um objeto de TProdutoAbstract e adiciona as informações deste produto no grid. Cada linha do grid representa um pedido. O grid também é usado como banco de dados da aplicação, uma vez que todos os pedidos lançados estão armazenados apenas no grid.

Não vou mostrar o código das classes TProdutoAbstract e suas implementações, pois o código é muito simples: não passa de dois pares de métodos Get e Set como aparece no diagrama de classes da **Figura 3**.

Nota do DevMan

Você não precisa necessariamente criar uma interface para respeitar o princípio "Programe para Interfaces". Ao invés de usar uma interface, você poderia usar classes abstratas. Dependendo da situação as classes abstratas são até melhores do que as interfaces. É o que acontece, por exemplo, no padrão Abstract Factory. Se quiser saber mais sobre como identificar quando é melhor usar uma interface ou uma classe abstrata, leia o artigo "POO – Crie sistemas flexíveis, expansíveis e com baixo custo de manutenção", na edição 110.

Nota

Já que a **Listagem 4** está usando as implementações de TProdutoAbstract, apenas porque as instâncias precisam ser criadas, talvez exista algum padrão criacional que ajude diminuir a dissidência criada na classe TForm1. Neste caso o Padrão Factory dá conta do recado.

Listagem 3. Configurando o TStringGrid do formulário

```
01  procedure TForm1.FormCreate(Sender:TObject);
02  begin
03   //Configura o título das colunas
04   StringGrid1.Cells[0,0] := 'Produto';
05   StringGrid1.Cells[1,0] := 'Preço';
06   StringGrid1.Cells[2,0] := 'Qtd.';
07   StringGrid1.Cells[3,0] := 'Total';
08   //Configura a largura de cada coluna
09   StringGrid1.ColWidths[0] := 200;
10   StringGrid1.ColWidths[1] := 80;
11   StringGrid1.ColWidths[2] := 40;
12   StringGrid1.ColWidths[3] := 80;
13  end;
```

Listagem 4. Criando um produto

```
01  procedure TForm1.BotaoAdicionarClick(Sender:TObject);
02  begin
03  case RadioGroupProdutos.ItemIndex of
04    //HAMBÚRGER
05    0:AdicionarProdutoAoGrid(THamburger.Create);
06    //BATATA FRITA
07    1:AdicionarProdutoAoGrid(TBatataFrita.Create);
08    //REFRIGERANTE
09    2:AdicionarProdutoAoGrid(TRefrigerante.Create);
10    //COMBO
11    3:
12    begin
13      AdicionarProdutoAoGrid(THamburger.Create);
14      AdicionarProdutoAoGrid(TBatataFrita.Create);
15      AdicionarProdutoAoGrid(TRefrigerante.Create);
16    end
17    else
18    begin
19      ShowMessage('Escolha um produto!');
20      Exit;
21    end;
22   end;
23  end;
```

Listagem 5. Adicionando um produto ao grid

```
01  procedure TForm1.AdicionarProdutoAoGrid
    (Produto:TProdutoAbstract);
02  var
03   UltimaLinha:Integer;
04  begin
05   //Verifica se a última linha do grid foi usada
06   if (StringGrid1.Cells[0,StringGrid1.RowCount - 1] <> '') then
07   begin
08    //Se a última linha do grid foi usada, adiciona mais uma linha
09    StringGrid1.RowCount := StringGrid1.RowCount + 1;
10   end;
11   UltimaLinha := StringGrid1.RowCount - 1;
12   StringGrid1.Cells[0,UltimaLinha] := Produto.GetDescricao;
13   StringGrid1.Cells[1,UltimaLinha] := FormatFloat
     ('R$ #,#0.00',Produto.GetPreco);
14   StringGrid1.Cells[2,UltimaLinha] := IntToStr(EditQuantidade.
     Value);
15   StringGrid1.Cells[3,UltimaLinha] := FormatFloat
     ('R$ #,#0.00',EditQuantidade.Value * Produto.GetPreco);
16  end;
```

Mesmo assim você poderá ver o código completo destas classes no arquivo de exemplo deste artigo, disponível para download. Agora que você já está familiarizado com o código do projeto, que comecem as alterações.

A exportação

Um novo requisito diz que o Sistema de Pedidos precisa exportar os pedidos cadastrados. Os pedidos devem ser exportados para um arquivo de texto puro (.TXT). O layout deste arquivo pode ser visto na **Listagem 6**.

As duas primeiras linhas do arquivo de exportação fazem parte do cabeçalho: na primeira linha é definida a data em que o arquivo foi gerado; na segunda linha é definida a quantidade de pedidos no arquivo. A seguir, entre as linhas 3 e 8 inclusive, está o corpo do arquivo. Nesta sessão estão todos os pedidos exportados – repare que cada pedido está relacionado a uma mesa. A última linha do arquivo deve possuir a string "Fim do arquivo" para indicar que o arquivo terminou.

Não parece ser difícil criar uma rotina que gere um arquivo como este layout. Algumas alterações com certeza serão necessárias, como por exemplo, adicionar o número da mesa em cada pedido lançado, mas convenhamos que isto não é complicado. No final das alterações o formulário do Sistema de Pedidos estará como aparece na **Figura 4**.

Vou começar as alterações adicionando o número da mesa que fez o pedido. Para isto, além de adicionar o TSpinEdit ao formulário também é necessário alterar algum código. Na **Listagem 3** está o método FormCreate, usado para configurar o grid quando o formulário é criado. Neste método configurei uma nova coluna no grid onde deve aparecer o número da mesa relacionada ao pedido. Na **Listagem 5** está o método AdicionarProdutoAoGrid usado para adicionar e popular uma linha do grid. Este método foi alterado para popular a nova coluna onde deve aparecer o número da mesa.

Tanto a nova coluna como o componente TSpinEdit podem ser vistos no segundo formulário da **Figura 4**. Não vou mostrar o código alterado, pois é simples e não está relacionado ao assunto principal do artigo.

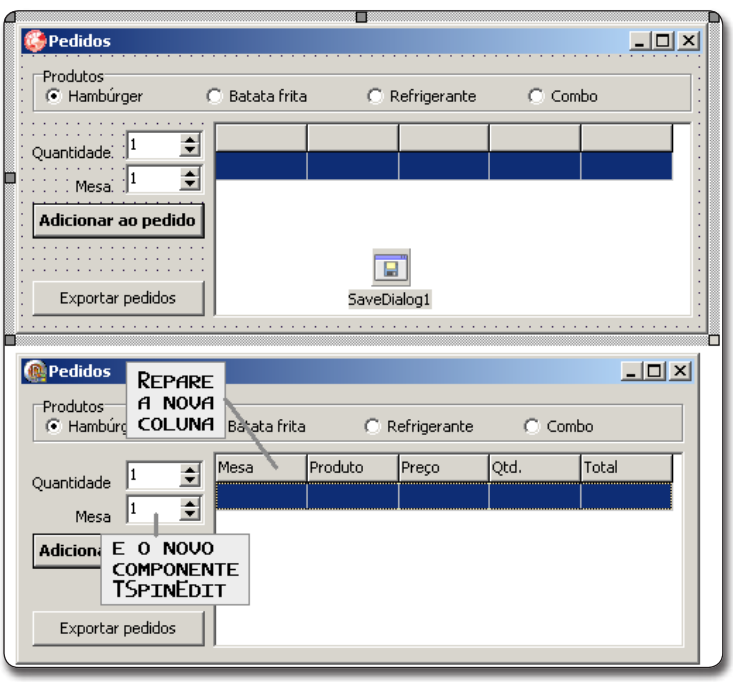


Figura 4. Sistema de Pedidos com a função de exportação

Agora que o Sistema de Pedidos está preparado para receber o número da mesa a exportação dos pedidos pode ser criada. Lembre-se que o suporte ao número da mesa foi adicionado porque o layout do arquivo de exportação exige esta informação.

O método de exportação

Vou criar um método chamado ExportarPedidos na classe TForm1. Ele será o responsável por gerar a exportação dos pedidos e salvar em um arquivo texto de acordo com o layout determinado. Apesar de ser um método extenso, ele não é complicado – veja **Listagem 7**.

Antes de começar a exportação eu preciso ter certeza de que existem pedidos a ser exportados. Entre as linhas 9 e 18 um loop determina a quantidade de pedidos e notifica o usuário caso não haja pedidos para exportar. Depois, entre as linhas 19 e 37 o arquivo de exportação é montado de acordo com o layout.

Nas linhas 20 e 21 o cabeçalho do arquivo é criado; entre as linhas 23 e 35 o corpo é criado; na linha 36 a string "Fim do arquivo" é adicionada indicando que o arquivo terminou; por fim, na linha 37, o arquivo é salvo no local determinado pelo parâmetro `destino`.

Sei que não expliquei muitos detalhes do código da **Listagem 7**, mais não se prenda a eles, porque são irrelevantes para o assunto principal deste artigo (Padrão Builder). Se mesmo assim, você quiser saber mais detalhes sobre o que está acontecendo nesta listagem, veja os comentários disponíveis no exemplo disponível para download.

Para finalizar, falta implementar o evento OnClick do botão "Exportar pedidos". Este evento mostra uma janela "Salvar como" para que o usuário escolha onde o arquivo de exportação será salvo. Veja **Listagem 8**.

Princípio KISS

Projetos orientados a objetos não devem ser complicados. O princípio KISS (Keep it Simple, Silly (ou Stupid) – Mantenha simples, bobo), é um princípio geral que defende a simplicidade de um projeto. Segundo o princípio KISS, toda complexidade desnecessária deve ser descartada.

Posso dizer que o método da **Listagem 7** com certeza é complicado e é possível simplificá-lo. Muitas modificações podem ser feitas neste método para torná-lo mais simples, mas vou escolher somente uma que irá simplificar todo o projeto e não apenas o método em si.

O Sistema de Pedidos é totalmente dependente das informações existentes no grid e de como estas informações aparecem nele. Se o grid for alterado, praticamente toda a aplicação também deverá ser alterada. Um exemplo disto aconteceu no tópico "A exportação", onde foi necessário criar uma nova coluna no grid para mostrar o número da mesa do pedido. O resultado disto foi uma alteração em todos os métodos mais importantes do projeto.

Como se isto já não fosse o suficiente, o método ExportarPedidos (**Listagem 7**), está "poluído" e complicado porque ele interage diretamente com o grid para buscar os pedidos existentes. Lembre-se que o grid é usado como se fosse o banco de dados da aplicação.

Para tornar o projeto mais simples, vou criar uma classe nova para representar os pedidos e assim métodos como o da **Listagem 7** podem interagir diretamente com os objetos desta nova classe ao invés de usar as informações do grid.

Listagem 6. Layout da exportação no formato TXT

```
01  02/01/2011
02  4 Pedidos
03  Mesa 1
04  Produto X, 2
05  Produto Y, 2
06  Produto Z, 2
07  Mesa 2
08  Produto H, 2
09  Fim do arquivo
```

Listagem 7. O método ExportarPedidos

```
01  procedure TForm1.ExportarPedidos(Destino: TFileName);
02  var
03    LinhaAtual, QuantidadeDePedidos: Integer;
04    NumeroDaMesa, Pedido: string;
05    ConteudoDoArquivo: TStrings;
06    PedidosPorMesa: TDictionary<string, TStrings>;
07  begin
08    QuantidadeDePedidos := 0;
09    for LinhaAtual := 1 to StringGrid1.RowCount - 1 do
10    begin
11      if StringGrid1.Cells[0, LinhaAtual] <> '' then
12        QuantidadeDePedidos := QuantidadeDePedidos + 1;
13    end;
14    if QuantidadeDePedidos = 0 then
15    begin
16      ShowMessage('Não existe pedido para exportar');
17      Exit;
18    end;
19    ConteudoDoArquivo := TStringList.Create;
20    ConteudoDoArquivo.Add(FormatDateTime('dd/mm/yyyy', Date));
21    ConteudoDoArquivo.Add(IntToStr(QuantidadeDePedidos)
      + ' pedido(s)');
22    PedidosPorMesa := TDictionary<string, TStrings>.Create
      (StringGrid1.RowCount);
23    for LinhaAtual := 1 to StringGrid1.RowCount - 1 do
24    begin
25      NumeroDaMesa := StringGrid1.Cells[0, LinhaAtual];
26      Pedido := StringGrid1.Cells[1, LinhaAtual] + ',' + StringGrid1.
        Cells[3, LinhaAtual];
27      if not PedidosPorMesa.ContainsKey(NumeroDaMesa) then
28        PedidosPorMesa.Add(NumeroDaMesa, TStringList.Create);
29      (PedidosPorMesa.Items[NumeroDaMesa] as TStrings).
        Add(Pedido);
30    end;
31    for NumeroDaMesa in PedidosPorMesa.Keys do
32    begin
33      ConteudoDoArquivo.Add('Mesa ' + NumeroDaMesa);
34      ConteudoDoArquivo.AddStrings(TStrings(PedidosPorMesa.Items
        [NumeroDaMesa]));
35    end;
36    ConteudoDoArquivo.Add('Fim do arquivo');
37    ConteudoDoArquivo.SaveToFile(Destino);
38  end;
```

Listagem 8. O evento OnClick do botão "Exportar pedidos"

```
01  procedure TForm1.BotaoExportarClick(Sender: TObject);
02  Begin
03    if SaveDialog1.Execute then
04      ExportarPedidos(SaveDialog1.FileName);
05  end;
```

Princípio KISS: TPedido

A **Figura 5** mostra o diagrama de classes do Sistema de Pedidos depois que a classe TPedido for criada. Como sempre um diagrama de classes revela muita informação – é como dizem: uma imagem vale mais que mil palavras.

A classe TPedido disponibiliza os métodos de acesso para seus atributos. Mas, repare dois detalhes importantes: no diagrama de classes a classe TPedido não mostra os métodos GetProduto e SetProduto e além disso esta classe possui um método GetTotal.

Nota do DevMan

O princípio de responsabilidade única defende que toda classe deve ter apenas uma única responsabilidade, ou seja, toda classe deve ter apenas uma única razão para sofrer alterações. Para saber mais sobre este princípio, leia o artigo "Padrão Strategy", na edição 126.

Nota

Nada de código desta vez! A classe TPedido possui apenas métodos Get e Set que não precisam ser mostrados para que você possa acompanhar o artigo. Lembre-se de que todos estes métodos estão disponíveis no arquivo para download.

Nota

Sei que você está ansioso para ver como implementar o Padrão Builder. Mas tenha paciência: saber quando e porque implementar um padrão é tão ou mais importante quanto saber implementá-lo. Lembre-se de que você precisa compreender o padrão e não decorá-lo.

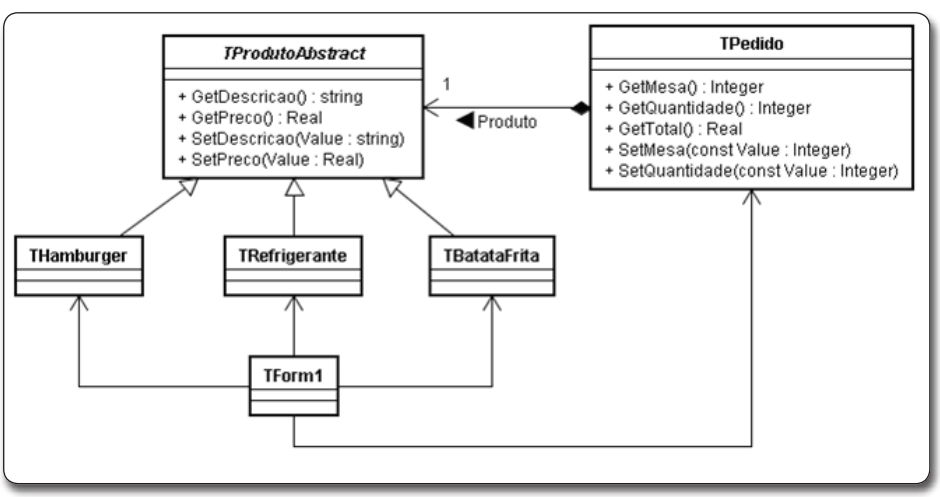


Figura 5. Classe TPedido

O método GetTotal vai calcular o valor total do pedido ao invés de deixar esta responsabilidade espalhada por todo o projeto como acontecia até agora. Os métodos GetProduto e SetProduto não aparecem no diagrama pois a linha que liga a classe TPedido e TProdutoAbstract, indica que TPedido é composto por TProdutoAbstract. Isto torna implícito que TPedido disponibiliza pelo menos um método de acesso para esta composição.

O diagrama ainda mostra mais uma informação importante: a classe TForm passou a depender de mais uma classe. Atualmente TForm1 é a classe mais problemática do projeto, pois ela depende de muitas outras classes para funcionar. TForm1 tem a necessidade de criar as instâncias das classes que ela deseja usar. Este é o típico problema resolvido pelos padrões criacionais.

Princípio KISS: TObjectList<TPedido>

Atualmente o grid usado no Sistema de Pedidos atua como uma espécie de banco de dados da aplicação, pois os pedidos são adicionados ao grid mas não são armazenados em nenhum outro local. Quando algum método (como o da **Listagem 7**) precisa interagir com os pedidos cadastrados é necessário criar um loop para ler cada linha do grid, pois é lá o único local onde os pedidos estão armazenados.

Para simplificar o projeto ainda mais, uma variável será declarada para armazenar a lista de pedidos lançados. Esta variável será o banco de dados da aplicação e o grid finalmente será usado apenas para representar a lista de pedidos – apenas REPRESENTAR e nada mais.

Está será uma alteração rápida e simples. Primeiro será declarada uma variável privada na classe TForm1 do tipo TObjectList<TPedido>. Veja como esta variável é declarada na **Listagem 9**. Em seguida o método FormCreate é alterado para criar esta variável. Atualmente este método é usado para configurar a aparência do grid e continuará sendo usado para este fim (**Listagem 3**). Na linha 3 da **Listagem 10** a variável ListaDePedidos é criada.

Agora que a classe TPedido existe, o método AdicionarProdutoAoGrid (**Listagem 5**), não tem mais sentido. Por isso este método será descartado (excluído), e no lugar dele será criado um novo método chamado AtualizarGrid. Este método servirá para atualizar as informações do grid de acordo com os pedidos existentes na variável ListaDePedidos – veja **Listagem 11**. Para finalizar esta sequência de alterações falta atualizar mais dois métodos: BotaoAdicionarClick (**Listagem 4**) e ExportarPedidos (**Listagem 7**).

As alterações feitas no método BotaoAdicionarClick não são importantes para o artigo, mas ele está disponível na **Listagem 12** caso queira dar uma olhada rápida em como ele ficou depois de atualizado. Por fim, o método ExportarPedidos também foi atualizado parar usar a variável ListaDePedidos ao invés das informações disponíveis no grid – o método ExportarPedidos atualizado está disponível na **Listagem 13**.

Com este mínimo de alteração o projeto ficou mais simples e menos "engessado". Os métodos mais importantes não dependem mais do tipo de componente usado para mostrar a lista de pedidos ao usuário. Além disto, é fácil observar como os métodos ficaram mais simples de entender, fazendo somente o que precisam – compare, por exemplo, a **Listagem 13** com a **Listagem 7** e você verá uma diferença para melhor.

Outro tipo de exportação (aqui começa o padrão Builder!)

Exportar os pedidos para um arquivo de texto puro já não é mais suficiente,

Listagem 9. Declarando a variável para lista de pedidos

```
01  TForm1 = class(TForm)
02    //Restante do código da classe TForm1
03  private
04    ListaDePedidos: TObjectList<TPedido>;
05  //Restante do código da classe TForm1
```

Listagem 10. Criando a lista de pedidos

```
01  procedure TForm1.FormCreate(Sender: TObject);
02  begin
03    ListaDePedidos := TObjectList<TPedido>.Create;
04    // Configura os títulos das colunas do grid
05    // Configura a largura de cada coluna do grid
06  end;
```

Listagem 11. Atualizando as informações do grid

```
01  procedure TForm1.AtualizarGrid;
02  var
03    Linha: Integer;
04    Pedido: TPedido;
05  begin
06    StringGrid1.RowCount := ListaDePedidos.Count + 1;
07    Linha := 0;
08    for Pedido in ListaDePedidos do
09    begin
10      Linha := Linha + 1;
11      StringGrid1.Cells[0, Linha] := IntToStr(Pedido.GetMesa);
12      StringGrid1.Cells[1, Linha] := Pedido.GetProduto.GetDescricao;
13      StringGrid1.Cells[2, Linha] := FormatFloat('R$ #,#0.00', Pedido.
        GetProduto.GetPreco);
14      StringGrid1.Cells[3, Linha] := IntToStr(Pedido.GetQuantidade);
15      StringGrid1.Cells[4, Linha] := FormatFloat('R$ #,#0.00', Pedido.GetTotal);
16    end;
17  end;
```

agora também é necessário exportar os pedidos para um arquivo no formato XML. O layout deste arquivo está disponível na **Listagem 14**.

Existem muitas maneiras de implementar este novo comportamento no Sistema de Pedidos. Uma delas com certeza é criar um novo método dentro da classe TForm1 com o objetivo de gerar a exportação no formato XML. Mas se isto for feito o resultado poderá ser desastroso com o passar do tempo.

Atualmente o usuário precisa de uma exportação para o formato TXT e XML, mas o que acontece quando for necessário um terceiro tipo de exportação? Será criado outro método dentro da classe TForm1? Isto seria péssimo, pois a classe TForm1 já tem muitas coisas para se responsabilizar e misturar muitas responsabilidades dentro de uma única classe não é algo "saudável" para um projeto.

Portanto, antes de implementar a exportação para o formato XML, a responsabilidade de montar os arquivos para exportação deve ser retirada da classe TForm1. A seguir veja como a classe TForm1 fica mais simples sem a responsabilidade de gerar o arquivo de exportação.

Simplificando ainda mais

Parece contraditório dizer que um novo comportamento será implementado no projeto e isto irá deixá-lo ainda mais simples.

Listagem 12. Método BotaoAdicionarClick atualizado

```
01  procedure TForm1.BotaoAdicionarClick(Sender: TObject);
02  Begin
03    case RadioGroupProdutos.ItemIndex of
04      0: CadastrarPedido(EditMesa.Value, THamburger.Create, EditQuantidade.Value);
05      1: CadastrarPedido(EditMesa.Value, TBatataFrita.Create, EditQuantidade.Value);
06      2: CadastrarPedido(EditMesa.Value, TRefrigerante.Create, EditQuantidade.Value);
07      3: // COMBO
08      Begin
09        CadastrarPedido(EditMesa.Value, THamburger.Create, EditQuantidade.Value);
10        CadastrarPedido(EditMesa.Value, TBatataFrita.Create, EditQuantidade.Value);
11        CadastrarPedido(EditMesa.Value, TRefrigerante.Create, EditQuantidade.Value);
12      End
13      Else
14        ShowMessage('Escolha um produto!');
15    end;
16  end;
17  procedure TForm1.CadastrarPedido(Mesa: Integer; Produto: TProdutoAbstract;
    Qtd: Integer);
18  var Pedido: TPedido;
19  begin
20    Pedido := TPedido.Create(Mesa, Produto, Qtd);
21    ListaDePedidos.Add(Pedido);
22    AtualizarGrid;
23  end;
```

Listagem 13. Método ExportarPedidos atualizado

```
01  procedure TForm1.ExportarPedidos(Destino: TFileName);
02  var
03    NumeroDaMesa: Integer;
04    Pedido: TPedido;
05    ConteudoDoArquivo: TStrings;
06    PedidosPorMesa: TDictionary<Integer, TStrings>;
07  begin
08    if ListaDePedidos.Count = 0 then
09    begin
10      ShowMessage('Não existe pedido para exportar');
11      Exit;
12    end;
13    ConteudoDoArquivo := TStringList.Create;
14    ConteudoDoArquivo.Add(FormatDateTime('dd/mm/yyyy', Date));
15    ConteudoDoArquivo.Add(IntToStr(ListaDePedidos.Count) + ' pedido(s)');
16    PedidosPorMesa := TDictionary<Integer, TStrings>.Create(StringGrid1.
      RowCount);
17    for Pedido in ListaDePedidos do
18    begin
19      if not PedidosPorMesa.ContainsKey(Pedido.GetMesa) then
20        PedidosPorMesa.Add(Pedido.GetMesa, TStringList.Create);
21      PedidosPorMesa.Items[Pedido.GetMesa].Add(Pedido.GetProduto
        .GetDescricao + ',' + IntToStr(Pedido.GetQuantidade));
22    end;
23    for NumeroDaMesa in PedidosPorMesa.Keys do
24    begin
25      ConteudoDoArquivo.Add('Mesa ' + IntToStr(NumeroDaMesa));
26      ConteudoDoArquivo.AddStrings(PedidosPorMesa.Items[NumeroDaMesa]);
27    end;
28    ConteudoDoArquivo.Add('Fim do arquivo');
29    ConteudoDoArquivo.SaveToFile(Destino);
30  end;
```

De fato isto acontecerá, pois algumas mudanças serão feitas para facilitar a implementação deste novo comportamento.

A exportação para o arquivo XML poderia ser implementada da forma como disse no tópico "Outro tipo de exportação", criando um novo método dentro da classe TForm1, mas isso tornaria o projeto mais complexo do que já está e um projeto flexível não pode ser complicado: ele deve ser simples.

Para simplificar, a responsabilidade de gerar os arquivos de exportação será retirada da classe TForm1, para ser colocada em outra classe cuja única responsabilidade é criar o arquivo de exportação. Assim o projeto começa a ficar de acordo com dois princípios: princípio KISS e princípio de responsabilidade única.

A **Figura 6** mostra a nova classe TExportaTxt cuja única responsabilidade é gerar o conteúdo dos arquivos de exportação. Ou seja, a classe TForm1 não sabe mais como os arquivos são gerados. A única coisa que TForm1 sabe é que precisa pedir para TExportaTxt gerar o conteúdo do arquivo TXT.

Para TExportaTxt gerar o conteúdo do arquivo de exportação ela precisa de algumas informações, como a data de criação do arquivo e os pedidos que serão exportados. Note que TExportaTxt poderia pegar os pedidos da variável ListaDePedidos existente na classe TForm, mais isto faria com que TExportaTxt dependesse da classe TForm1: isto é justamente o que precisa ser evitado.

Veja na **Listagem 15** como o método ExportarPedidos ficou mais simples. Para comparação veja como este método era nas **Listagens 7** e **13**: o método ficou simples e fácil de ler. Agora, quero destacar alguns pontos importantes.

Listagem 14. Layout da exportação no formato XML

```
01  <pedidos totalDePedidos="4" dataCriacao=
      "01/02/2011">
02   <mesa número="1">
03    <produto>
04     <nome>Produto 1</nome>
05     <quantidade>2</quantidade>
06    </produto>
07   </mesa>
08   <mesa número="1">
09    <produto>
10     <nome>Produto 2</nome>
11     <quantidade>2</quantidade>
12    </produto>
13   </mesa>
14   <mesa número="1">
15    <produto>
16     <nome>Produto 3</nome>
17     <quantidade>2</quantidade>
18    </produto>
19   </mesa>
20   <mesa número="2">
21    <produto>
22     <nome>Produto 1</nome>
23     <quantidade>1</quantidade>
24    </produto>
25   </mesa>
26  </pedidos>
```

Nota

O código da classe TExportaTxt não foi mostrado aqui propositalmente, pois não é importante saber como ela funciona, e sim como ela é usada. Claro que você pode baixar o arquivo de exemplo deste artigo e ver como tudo funciona dentro da classe TExportaTxt.

Nota

Alguns elementos não aparecem no diagrama de classes da **Figura 7**, pois não foram alteradas e não são importantes neste momento – você pode visualizar estas classes na **Figura 6**.

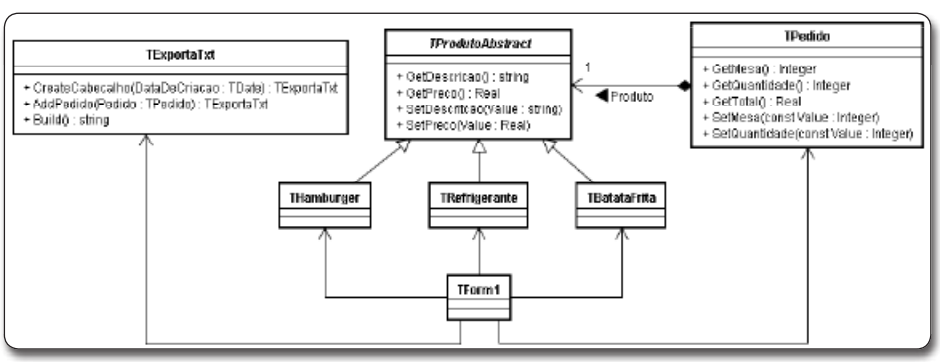


Figura 6. Classe TExportaTxt

Repare na linha 5, a declaração da variável Builder do tipo TExportaTxt. Apesar do nome o Padrão Builder não está implementado (ainda), mas está a caminho. Escolhi este nome para esta variável porque de fato ela é usada para construir o conteúdo do arquivo de exportação parte por parte. Você pode ver os métodos de construção CreateCabecalho e AddPedido sendo invocado nas linhas 13 e 15 respectivamente.

Nem você e nem a classe TForm1 sabe o que acontece quando estes métodos são invocados, mas TForm1 sabe que para definir a data de criação e para adicionar os pedidos que serão exportados, estes dois métodos devem ser usados.

Por fim na linha 20, o método Build é invocado. Este método retorna todo o conteúdo do arquivo de exportação. Como este conteúdo é criado, não é responsabilidade da classe TForm1. Tudo que TForm1 faz é salvar o retorno deste método dentro de um arquivo de texto puro. Ficou simples, não?

Antes de continuar, repare na **Figura 6** que os métodos CreateCabecalho e AddPedido retornam um objeto da mesma classe. Esta é uma prática conhecida como Interface Fluente, e permite que o programador crie instruções como esta:

```
Exportador := TExportaTxt.Create;
Exportador.CreateCabecalho(Date).AddPedido(P1).
AddPedido(P2).Build();
```

Veja que é possível encadear as chamadas dos métodos em uma única linha, pois cada um dos métodos retorna o mesmo objeto usado para invocá-lo, provendo assim uma interface fluente.

Agora o projeto está pronto para implementar a exportação em XML (**Figura 6**).

Exportando para XML

Para continuar respeitando o princípio de responsabilidade única, a responsabilidade de gerar o conteúdo do arquivo XML deve ficar separada em uma nova classe, sendo assim uma nova classe chamada TExportaXml será criada. Ela terá os mesmos métodos e também será usada da mesma maneira que TExportaTxt é usada. Observe a nova classe TExportaXml na **Figura 7**.

Provavelmente você notou que além de criar uma nova classe chamada TExportaXml, também foi definida uma interface chamada IExportacao, a qual é implementada pelas classes TExportaTxt e TExportaXml. Esta interface foi criada, pois como foi dito no último parágrafo, as classes TExportaTxt e TExportaXml terão os mesmo métodos e serão usadas da mesma forma, ou seja, elas possuem a mesma interface.

Mas uma vez repare que TForm1 não está respeitando o princípio programe para interfaces, pois ele depende das implementações de IExportacao. Para terminar de implementar a exportação para XML está faltando apenas atualizar o código dos métodos ExportarPedidos e BotaoExportarClick, bem como disponibilizar uma forma do usuário escolher qual formato de exportação ele deseja usar.

Para que o usuário possa escolher o formato de exportação um TComboBox está disponível logo abaixo do botão "Exportar pedidos" como pode ser visto na **Figura 8**. Veja o código do método BotaoExportarClick na **Listagem 16** e do ExportarPedidos na **Listagem 17**.

As alterações feitas no método BotaoExportarClick são evidentes e dispensam explicações. Já no método ExportarPedidos existem, pontos importantes. Primeiro, repare que o tipo da variável Builder, declarado na linha 5 foi alterada de TExportaTxt para IExportacao, ou seja, agora esta variável pode receber uma instância de qualquer uma das duas classes que a implementam IExportacao. A segunda alteração importante pode ser vista entre as linhas 12 e 15 da **Listagem 17**. É nestas linhas que a aplicação decide qual das duas classes de exportação usar de acordo com o formato de arquivo escolhido pelo usuário. O restante permanece igual.

Você reparou como foi simples implementar este novo comportamento. Em poucos parágrafos foi possível implementar um novo tipo de exportação totalmente diferente do que existia. Isto foi possível graças às mudanças feitas para simplificar o projeto e também graças ao Padrão Builder.

Isto mesmo! Graças ao Padrão Builder. Você não reparou que ele já foi implementado? Então veja a seguir.

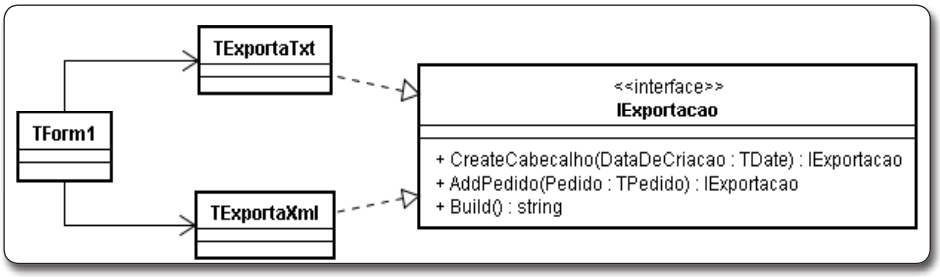


Figura 7. Classe TExportaXml

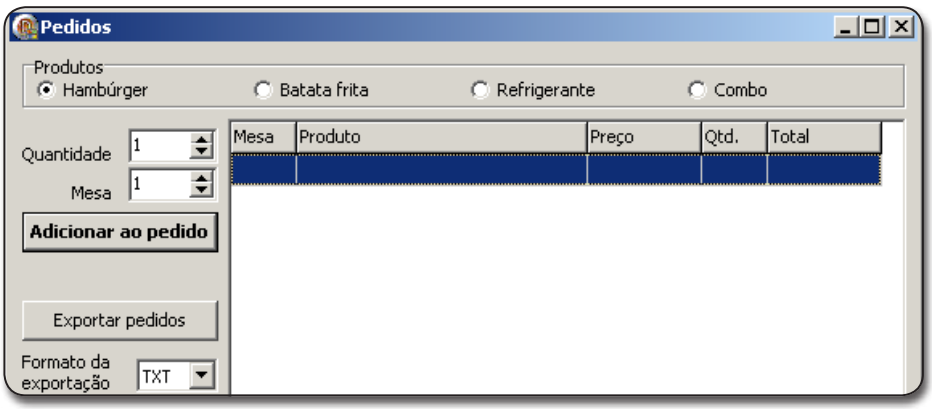


Figura 8. Versão final do Sistema de Pedidos

Listagem 15. Novo método de exportação

```
01  procedure TForm1.ExportarPedidos(Destino:
    TFileName);
02  var
03    Pedido: TPedido;
04    Arquivo: TextFile;
05    Builder: TExportaTxt;
06    begin
07    if GetListaDePedidos.Count = 0 then
08    begin
09      ShowMessage('Não existe pedido para
        exportar');
10      Exit;
11    end;
12    Builder := TExportaTxt.Create;
13    Builder.CreateCabecalho(Date);
14    for Pedido in GetListaDePedidos do
15      Builder.AddPedido(Pedido);
16    if FileExists(Destino) then
17      DeleteFile(Destino);
18    AssignFile(Arquivo, Destino);
19    Rewrite(Arquivo);
20    Write(Arquivo, Builder.Build);
21    CloseFile(Arquivo);
22  end;
```

O Padrão Builder revelado

É possível que você não tenha percebido, mas você já aprendeu como implementar o Padrão Builder. Vou explicar como. Lembre-se de que é necessário interpretar um padrão e identificar as classes do projeto que substituem os participantes do padrão.

Listagem 16. Método BotaoExportarClick atualizado

```
01  procedure TForm1.BotaoExportarClick
    (Sender: TObject);
02  Begin
03    if ComboFormatoExportacao.ItemIndex = 0
      then
04      SaveDialog1.Filter := '*.txt|*.txt'
05    Else
06      SaveDialog1.Filter := '*.xml|*.xml';
07    if SaveDialog1.Execute then
08      ExportarPedidos(SaveDialog1.FileName);
09  end;
```

Listagem 17. Método ExportarPedidos atualizado

```
01  procedure TForm1.ExportarPedidos
    (Destino: TFileName);
02  var
03    Pedido: TPedido;
04    Arquivo: TextFile;
05    Builder: IExportacao;
06  begin
07    if GetListaDePedidos.Count = 0 then
08    begin
09      ShowMessage('Não existe pedido para
        exportar');
10      Exit;
11    end;
12    if ComboFormatoExportacao.ItemIndex = 0
      then
13      Builder := TExportaTxt.Create
14    else
15      Builder := TExportaXml.Create;
16    Builder.CreateCabecalho(Date);
17    for Pedido in GetListaDePedidos do
18      Builder.AddPedido(Pedido);
19    if FileExists(Destino) then
20      DeleteFile(Destino);
21    AssignFile(Arquivo, Destino);
22    Rewrite(Arquivo);
23    Write(Arquivo, Builder.Build);
24    CloseFile(Arquivo);
25  end;
```

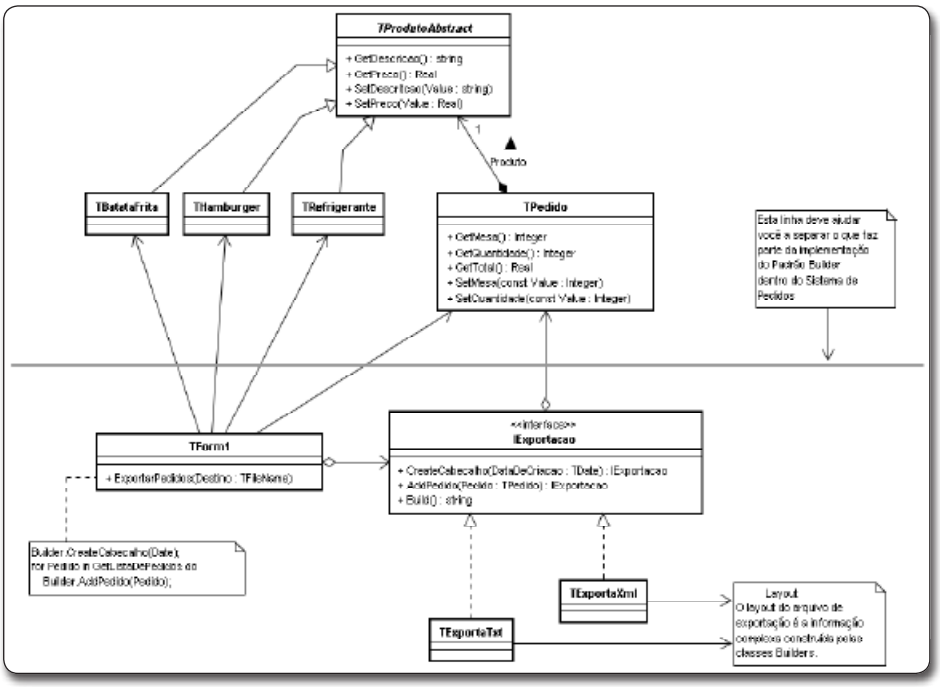


Figura 9. O Padrão Builder revelado

Um dos participantes do Padrão Builder, é a interface IBuilder, como você pode ver na **Figura 1**. Esta é a interface que declara todos os métodos necessários para construir o objeto complexo. Já o participante TConcreteBuilder representa as verdadeiras classes Builders que sabem como construir o objeto complexo. Embora no diagrama do Padrão Builder apareça apenas uma classe concreta não significa que apenas uma vai existir no projeto. Na verdade podem existir varias formas de construir um objeto complexo. Quando isto acontece existirão várias classes que implementarão a interface IBuilder.

No Sistema de Pedidos, acontece exatamente isto, pois o conteúdo do arquivo de exportação pode ser construído de pelo menos duas formas diferentes, então foram criadas duas classes chamadas TExportaTxt e TExportaXml, cada uma capaz de construir um layout para exportação. Estas duas classes são representadas pelo participante TConcreteBuilder do diagrama de classes do Padrão Builder. O participante IBuilder, representa a interface IExportacao no Sistema de Pedidos.

Obviamente TDirector representa a classe TForm1 do Sistema de Pedidos, pois é TForm1 quem usa o Padrão Builder. Lembre-se que podem existir mais de uma classe usando o Padrão.

Já o participante TProduct é a grande incógnita para identificar o Padrão Builder no Sistema de Pedidos, pois não existe uma classe no Sistema de Pedidos que corresponda ao participante TProduct. O fato é que o "produto" construído pelas classes construtoras é na verdade o layout de exportação. Claro que este layout poderia ser convertido em outra classe chamada TLayout, mas se isto fosse feito o artigo perderia o foco. Então, o participante TProduct representa o layout de exortação no Sistema de pedidos.

Para que não restem dúvidas, na **Figura 9** está disponível o diagrama de classes completo do Sistema de Pedidos. Este diagrama está divido ao meio por uma linha cinza. Compare a parte inferior do diagrama com a **Figura 1**. Consegue ver o Padrão Builder dentro do Sistema de Pedidos?

Conclusão

A complexidade do Sistema de Pedidos aumentou pouco a pouco conforme as alterações chegavam. Em certas ocasiões a melhor coisa foi reavaliar as decisões tomadas no passado. Um exemplo disto acontece no tópico "Princípio KISS: TObjectList<TPedido>", onde o projeto alcança certo nível de complexidade e algumas decisões tomadas anteriormente começam a não fazer mais sentido.

A partir disto uma série de alterações foi feitas antes de continuar com as alterações pedidas pelo usuário. Na vida real, esta é uma situação comum, pois você não deve ter medo de voltar atrás e alterar coisas que faziam sentido quando foram criadas, mas não fazem sentido agora. Sempre avalie seu código como se ele tivesse sido escrito por outro programador.

Por fim, chega um momento que a responsabilidade de criar o conteúdo do arquivo de exportação precisa ser separada. Além disto, criar o conteúdo do arquivo não é um processo simples que pode ser executado com apenas uma instrução. Para criar o conteúdo do arquivo, são necessárias várias informações, como data de criação do arquivo e a lista de pedidos que será exportada: ou seja, criar o conteúdo do arquivo é um processo complexo.

O Padrão Builder "serviu como uma luva" para este problema. Com ele foi possível simplificar a construção do conteúdo dos arquivos de exportação.

Rafael Stavarengo
faelsta@gmail.com
http://stavarengo.com
Programador de sistemas da Cheina Informática com 9 anos de experiência, integrante da equipe editorial da revista ClubeDelphi. Domínio em Java, PHP e UML. Sólido conhecimento em Design Patterns e metodologias ágeis. Graduado em Análise e Desenvolvimento de Sistemas pela UNIPAR.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Visual Studio vs. RAD Studio

Comparativo entre os IDEs, para desenvolvedores Delphi Prism – Parte 3

Rodrigo Araujo

Resumo DevMan

De que trata o artigo:

O artigo irá demonstrar algumas funcionalidades relacionadas à produção de sistemas utilizando o Delphi Prism, auxiliando no que diz respeito ao ambiente de desenvolvimento, dando ênfase ao relacionamento em comum de tarefas realizadas no Delphi XE que se assemelham no Delphi Prism, dessa forma, familiarizando aqueles que não estão adaptados ao IDE do Visual Studio.

Para que serve:

Serve como uma forma de dar as primeiras impressões sobre o ambiente de desenvolvimento do Prism realizando destaques visuais relacionados à programação e objetivados no que há de diferente nas ferramentas, menus e janelas para facilitar e auxiliar a produção de projetos, assim diminuindo impactos ao iniciar a desenvolver neste ambiente.

Em que situação o tema é útil:

Este artigo visa ajudar na produção de sistemas com o Delphi Prism, utilizando o ambiente integrado do Visual Studio. Ao demonstrar suas principais funcionalidades trará conhecimento suficiente para quem está se adaptando a este ambiente, principalmente a quem utiliza outras ferramentas como o Delphi XE, facilitando e dando condições iniciais para o desenvolvimento.

Visual Studio vs. RAD Studio:

O artigo dará destaque às opções visuais e de IDE que irão auxiliar o desenvolvimento de sistemas. Seguindo a demonstração de opções que facilitarão a programação, surgem menus como o menu auxiliar, ao clicar com o botão direito do mouse em uma área de código, a pesquisa de itens, em busca de informações ou trechos de código. O menu Project e suas vastas opções e configurações para o desenvolvimento, o menu tools apresentando maneiras de realizar tarefas através de ferramentas auxiliares. A janela de Error List indicando os erros do aplicativo ao realizar uma compilação, a formatação e design de código, permitindo estilos mais limpos e elegantes. A utilização da Task List possibilitando adicionar mensagens para tarefas a serem produzidas. O uso de uma funcionalidade de Quick Access auxiliando e facilitando através do teclado a utilizar opções de menu de forma rápida, as ferramentas para Code Analysis que ajudam a mensurar a qualidade e produtividade de um código fonte e, por fim, o menu data que permite realizar acesso e manutenção a fontes de dados distintas.

Ao utilizar o Delphi Prism, na janela de código, o menu acessado através do botão direito do mouse para o Visual Studio 2010, possui algumas opções interessantes, conforme está descrito na **Figura 1**. Este menu possibilita realizar algumas operações básicas relacionadas, principalmente a geração de trechos de códigos. Entre os destaques está presente a opção View Designer, onde é realizada a transição da janela de código para a janela de designer, opção também disponível por atalho de teclado, como pode ser visto na imagem anterior, pelas teclas Shift + F7.

A possibilidade de Insert Snippet permitindo adicionar blocos de códigos pré-definidos com funcionalidades específicas facilitando e acelerando o desenvolvimento de códigos. A busca e localização precisa para a definição de determinada palavra no código, como uma classe, um método ou uma variável, opção esta presente em Go To Definition e sua tecla de atalho F12. As opções para auxiliar na criação de classes, variáveis e métodos disponíveis em Create Class, Create Variable e Create Method. O uso de uma função, que poderá ser bastante útil, através da opção Paste C# as Oxygene, nela é possível copiar um código escrito na linguagem C# e colar diretamente para o Delphi Prism, assim, o código colado será traduzido para a linguagem do Prism.

Além disso, existem outras opções como a adição de breakpoints, os tradicionais copiar e colar e a formatação de código disponível na opção de menu Outlining auxiliando na melhor forma de visualização de código, expandindo ou contraindo a visualização de determinados trechos, principalmente para métodos e classes.

Ao utilizar o Delphi XE, o menu da **Figura 2**, também disponível através do botão direito do mouse na área de código, demonstra quais as opções estão disponíveis para o tratamento de código na programação de um aplicativo. Existem várias funcionalidades disponíveis para a programação, entre elas as formas comuns de copiar e colar, fechar a página, abrir uma nova janela, pesquisar, opção de ajuda, complete class, semelhante ao que ocorre no Prism. Outras opções estão presentes como a adição de marcadores do tipo Bookmark, depuração de código pelo menu Debug, visualização de mensagens, adição de comentários e formatação de código através da opção Format Source, deixando o código mais visualmente elegante.

Pesquisa de itens

A pesquisa no Delphi Prism é simplificada e aprimorada, sendo possível pesquisar utilizando as teclas Ctrl em conjunto com a tecla , (vírgula), visualizada na **Figura 3**.

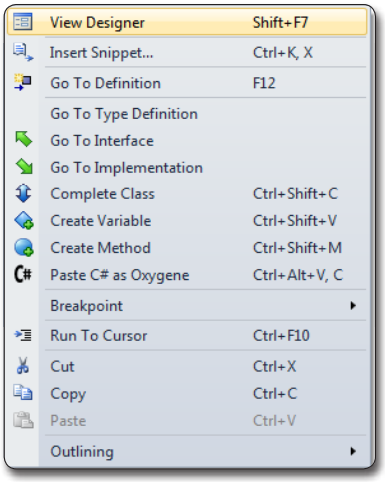


Figura 1. Menu auxiliar para utilização no Delphi Prism

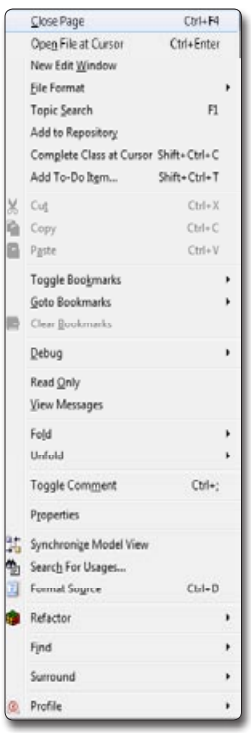


Figura 2. Menu auxiliar para utilização no Delphi XE

Algumas características podem ser observadas para a janela Navigate To, entre elas, se a sequência de pesquisa contém caracteres maiúsculos, a busca será case-sensitive, caso contrário não. O espaço em branco entre os caracteres será tratado como um relacionamento "and" para as palavras a serem pesquisadas. A pesquisa não oferece suporte a caracteres curinga, assim, esta característica torna-se extremamente útil quando o projeto em desenvolvimento é complexo e, mesmo assim, o resultado da pesquisa é apresentado de forma rápida.

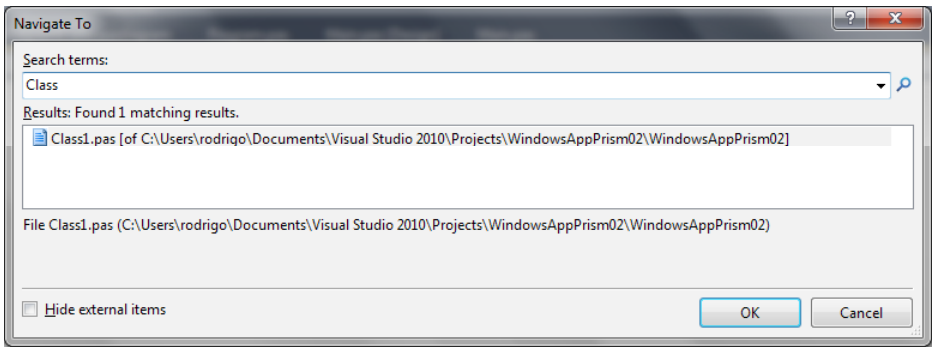


Figura 3. Menu para realizar a pesquisa de itens no Prism

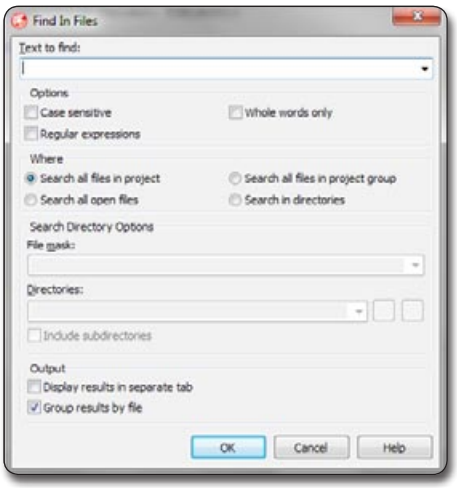


Figura 4. Menu para realizar a pesquisa de itens no Delphi XE

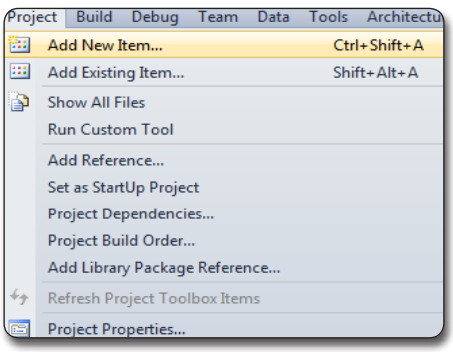


Figura 5. Menu Project

De maneira semelhante, o Delphi XE apresenta a pesquisa simplificada e a pesquisa estendida através da janela demonstrada na **Figura 4**. O exemplo indica a pesquisa de itens que será realizada nos arquivos disponíveis no projeto que está sendo desenvolvido, semelhante ao que ocorre com o Delphi Prism.

Menu Project

O menu de projetos do Delphi Prism apresenta algumas possibilidades de utilização conforme pode ser visualizado na **Figura 5**.

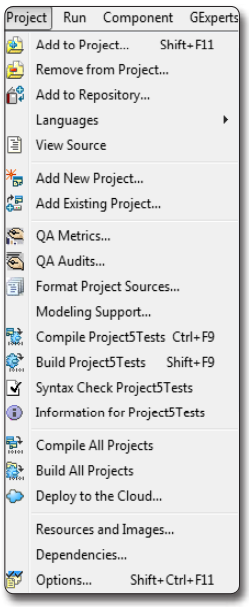


Figura 6. Menu Project presente no Delphi XE

Este menu apresenta opções de adição de novos itens ao projeto já existente, adição de itens previamente desenvolvidos e visualização de todos os arquivos do projeto. Outras opções também estão disponíveis como a adição de referências, indicação do projeto inicial a ser executado, as dependências de projeto e um menu auxiliar sobre as propriedades do projeto.

A característica básica de Set as Startup Project determina que um projeto será o primeiro a ser construído, depurado, analisado e executado quando a solução de projetos for implantada. Através do menu de projetos ou da Solution Explorer poderá ser indicado qual será o projeto inicial. Essa funcionalidade é bastante interessante, principalmente quando vários projetos fazem parte da mesma solução, e cada um apresenta regras e funcionalidades próprias,

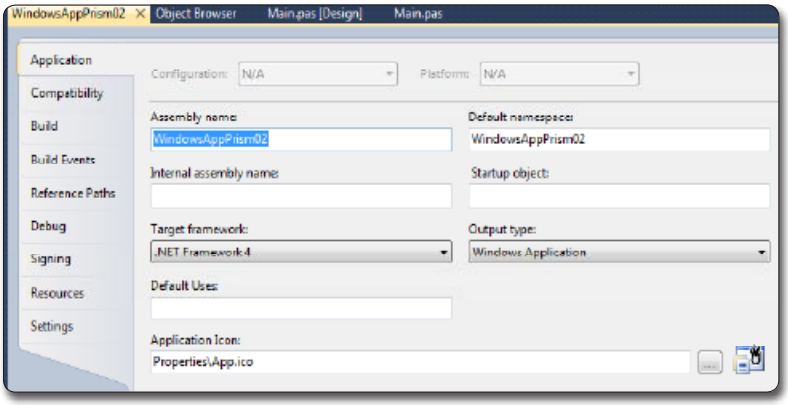


Figura 7. Menu com opções de Projeto no Prism

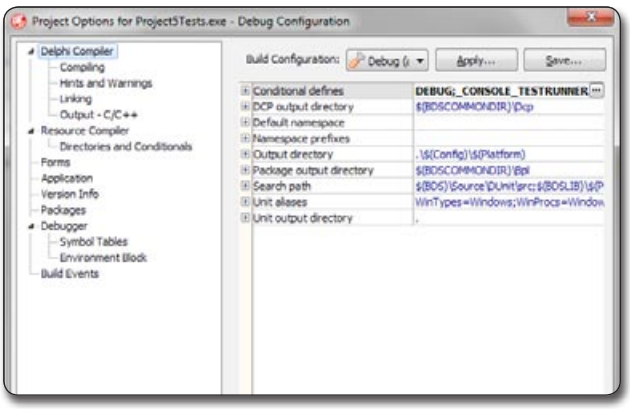


Figura 8. Menu com opções de Projeto para o Delphi XE

que individualmente apresentam comportamentos distintos, mas em conjunto com interfaces e páginas web constituem toda a representação de uma solução.

A **Figura 6** indica o mesmo menu de projeto, mas utilizado no Delphi XE com algumas outras opções disponíveis. Para o Delphi XE existem opções semelhantes ao Prism como adição de itens ao projeto, remoção, adição de novos projetos, opções de definição do projeto e informações. Outras funcionalidades direcionadas ao XE são apresentadas como compilar o projeto, recursos e imagens, dependências, suporte a modelagem, métricas e gerar um executável de todos os projetos.

Além disso, as opções de projeto podem ser acessadas, visualizadas e modificadas através do menu de projetos e descritas conforme a **Figura 7** no Delphi Prism. As opções de projeto disponíveis no Delphi Prism são várias e incluem definições de configuração para a aplicação, incluindo versão do .NET Framework a ser utilizada, o tipo de aplicação, entre elas console, Windows ou uma biblioteca de classes. Outras funcionalidades relacionadas com a depuração, recursos, referências, compatibilidade e geração do aplicativo também podem ser observadas.

De forma semelhante, o Delphi XE apresenta uma janela com as opções de projeto semelhantes à descrita anteriormente, mas com características próprias do Delphi que podem ser vistas na **Figura 8**. No Delphi XE as opções de projeto apresentam funcionalidades semelhantes ao Prism, com disposições diferentes e características próprias. Entre elas, as opções de compilador, recursos para o compilador, inclusive a indicação de diretórios, versionamento, pacotes, opções de depuração e geração do aplicativo.

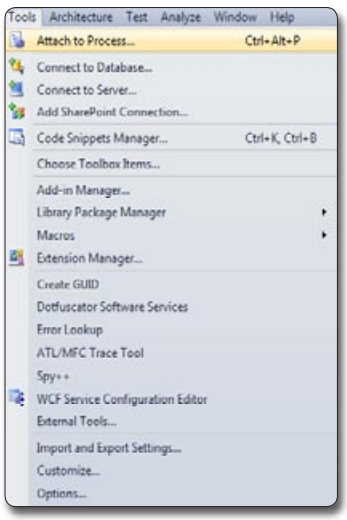


Figura 9. Menu Tools para o Delphi Prism

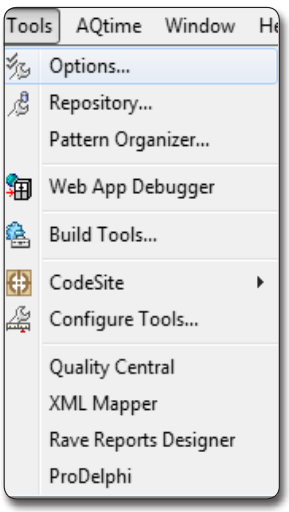


Figura 10. Menu Tools para o Delphi XE

Menu Tools

Neste menu, descrito na **Figura 9**, são utilizadas opções para conexão a banco de dados, servidores e SharePoint, gerenciamento de controles da janela toolbox, code snippets, macros e gerenciador de extensões presentes no Visual Studio. Outras opções como criação de GUID, configurações de ferramentas externas, opções de ferramentas e importação e exportação das configurações utilizadas ou modificadas também estão aqui descritas.

Além disso, existe a opção de definição de Macros, bastante úteis para desenvolver tarefas rotineiras, que sempre ocorrem e que devem ser realizadas. O uso de Spy++ possibilita realizar o entendimento do layout de janelas de um aplicativo em execução ou identificar mensagens que podem ocasionar erros de depuração. Com ele a visualização de eventos e valores de propriedades aparece bem visível para análise. A opção de menu chamada de ATL/MFC descreve ferramentas de monitoramento para classes compartilhadas auxiliando na programação.

Ao utilizar o Delphi XE, o menu Tools é apresentado conforme a **Figura 10** com algumas opções para utilização em projetos e ferramentas que auxiliam no desenvolvimento.

As opções do menu tools possibilitam realizar configurações de ambiente, como o editor irá se comportar, configurações da paleta, cores, toolbar, variáveis, html, websnap, maneiras de visualização, mapeamento de teclado e código fonte. Configurações e gerenciamento do repositório de objetos incluindo os projetos e seus arquivos. Ferramentas para geração do aplicativo, suas configurações, ferramentas externas como CodeSite, mapeamento

de documentos XML e design para utilização de RAVE Reports.

Uma das principais ferramentas que merece extremo destaque é o CodeSite que auxilia com formas avançadas de depuração, através da adição de mensagens em determinados fluxos de códigos de uma aplicação, possibilitando verificar o comportamento do aplicativo. Ele fornece informações mais detalhadas de como o código está sendo executado, permitindo localizar problemas de uma maneira mais ágil. Assim, dará maior segurança para que o aplicativo seja executado corretamente, permitindo de uma forma eficaz a realização do ciclo de vida do aplicativo. Possui algumas características principais como a não interrupção do fluxo de execução da aplicação, as suas classes de logs podem capturar vários tipos de informações, os dados de estruturas complexas são facilmente armazenados, possui fácil configuração e permite agrupar os logs em grupos definidos pelo usuário.

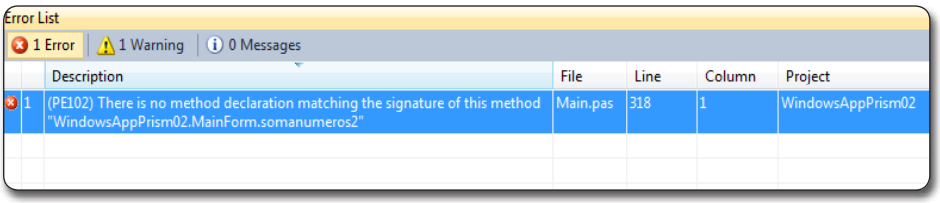


Figura 11. Janela de erros no Delphi Prism

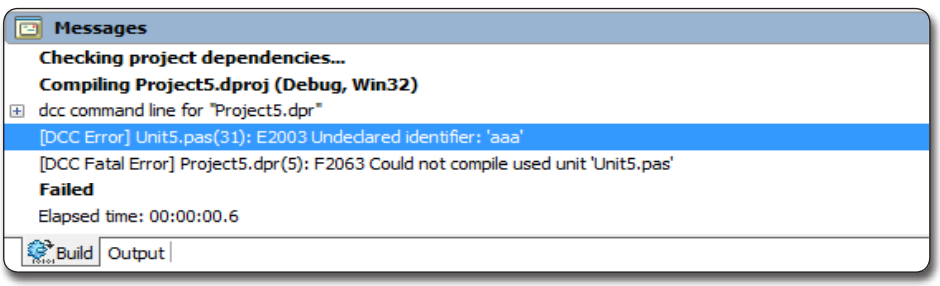


Figura 12. Janela de mensagens no Delphi XE

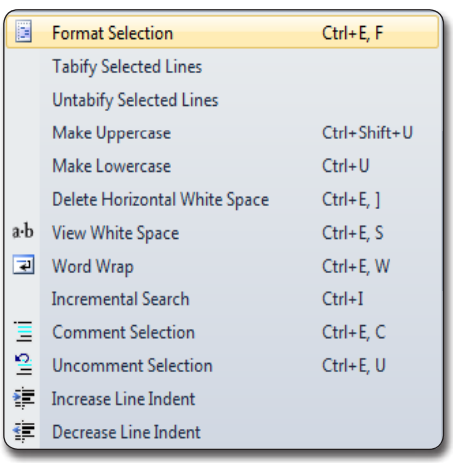


Figura 13. Menu para formatação de código

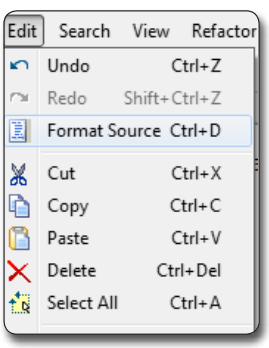


Figura 14. Menu para formatação de código no Delphi XE

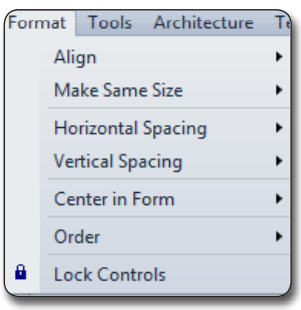


Figura 15. Menu para auxilar no Design

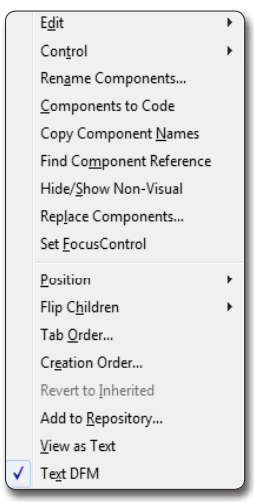


Figura 16. Menu para auxilar no Design do Delphi XE

Error List

A janela de lista de erros, observada na **Figura 11**, auxilia no desenvolvimento de aplicações, nesta janela de erros é possível visualizar os erros, warnings (avisos) e mensagens produzidas ao compilar um código de aplicativo, encontrar erros de sintaxe, erros de implantação e erros de análise. Ao realizar dois cliques em qualquer entrada de mensagem de erro o arquivo contendo o código fonte onde o problema está situado, será localizado, indicando sua linha e sua coluna para uma informação mais precisa do que será necessário modificar.

Além disso, permite filtragem de dados e organização de colunas a serem visualizadas na janela de erros. A lista pode ser classificada de acordo com a coluna que for selecionada e, para reclassificar basta clicar em outra coluna ou para alterar a ordem em que as colunas serão exibidas arraste qualquer cabeçalho de coluna para a esquerda ou direita.

Na janela descrita são visualizadas as colunas de erros, warning e messages, possibilitando observar os erros presentes no projeto, avisos e mensagens, além da indicação de qual a descrição do erro, arquivo, linha e coluna, facilitando a análise e depuração do código. Neste caso, a

Nota

ATL significa Active Template Library e representa um conjunto de classes baseadas em templates. MFC significa Microsoft Foundation Class e indicam um conjunto de componentes reutilizáveis que encapsulam funções básicas para o desenvolvimento de aplicativos.

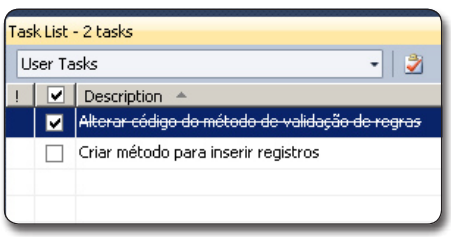


Figura 17. Uso de Task List

aba selecionada é a de erros, a segunda representa os avisos através de Warning, sendo descrita de forma semelhante à aba de erros, e, por último, aparece a aba de mensagens indicando mensagens informativas, que não irão interferir no processo de execução do aplicativo.

A **Figura 12** demonstra a janela de mensagens utilizadas no Delphi XE no momento de compilação de um projeto, nesta janela são visualizadas as informações de compilação e conterá as descrições dos erros, caso existam. Na janela são visualizadas as opções de compilação do projeto, verificações de dependências, indicação de tempo decorrido e as mensagens de erro e warning, caso existam, ao ocorrer um erro, como foi descrito, ao realizar dois cliques na linha, o Delphi XE irá indicar diretamente o arquivo e a linha em que o problema de compilação está presente.

Formatação de código

Através do menu Edit na opção Advanced está presente o menu da **Figura 13** que possibilita realizar operações de formatação de código como comentar linhas, descomentar, aumentar a endentação e diminuir. Além disso, uma funcionalidade bastante interessante presente no Visual Studio é a utilização da autoformatação de código, dessa forma, simplesmente é necessário ir digitando o código que o sistema irá formatar de maneira adequada todos os seus itens.

Para o Delphi XE, opção semelhante está disposta no menu Edit onde através da **Figura 14** é demonstrada a opção para formatação do código fonte. A opção de menu Format Source permite realizar uma formatação em todo o código fonte do projeto, realizando os ajustes necessários de acordo com as configurações presentes no Delphi XE e organizando de forma mais clara os códigos presentes. Além disso, também possui as tradicionais endentações de regiões de código, ao selecionar determinados trechos de código, é possível em combinação com as teclas Control + Shift + i e Control + Shift + u aumentar ou diminuir a endentação do código.

Design e formatação

Ao utilizar o modo de design no Visual Studio é possível realizar a formatação dos controles presentes no design, redimensioná-los, trocar ordens e modificar alinhamentos, dessa forma, o menu Format apresenta opções que irão auxiliar nesta tarefa, a **Figura 15** demonstra as alternativas presentes. O menu de formatação permite alterar os alinhamentos, espaçamentos horizontais e verticais, centralizações relacionadas ao formulário, ordem e bloqueio de controles.

De forma semelhante, no Delphi XE a **Figura 16** mostra as opções à disposição de uso. Estão presentes no menu opções para realizar tarefas de design, possibilitando modificar as posições, tab order, substituição de componentes, foco entre outras disponíveis conforme pode ser observado.

Task List

A Task List auxilia o programador a criar uma lista de tarefas a serem realizadas. Estas tarefas poderão ser visualizadas em uma janela e são divididas em dois tipos, as User Tasks que são anotações sobre o trabalho a ser realizado e as Task Comments que são anotações realizadas através de comentários diretamente no código fonte indicando onde o trabalho deve ser realizado.

Para visualizar esta janela basta acessar o menu View e clicar na opção Task List, nesta janela, as User Tasks são adicionadas diretamente na janela, como pode ser visualizado na **Figura 17**, enquanto os Comments são analisados diretamente no código através da adição de comentários do tipo // TODO, HACK ou UNDONE como será observado mais adiante na **Figura 18**.

Na **Figura 17** foi selecionada a opção de User Tasks na lista superior, realizando o filtro somente por tarefas adicionadas diretamente na janela. Dessa forma, foram adicionadas duas tarefas, a primeira, que já foi concluída, apresenta-se riscada e a segunda incompleta demonstra-se de forma diferente. Esta é uma maneira simples de se realizar anotações para o projeto.

A **Figura 18** demonstra o uso de comentários diretamente no código no local onde

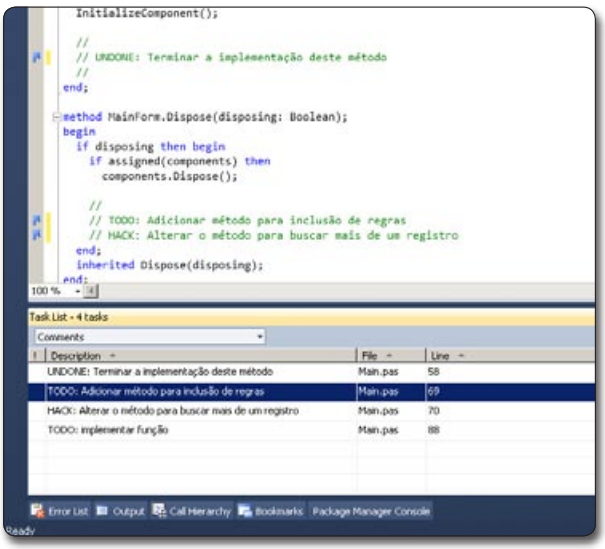


Figura 18. Uso de Task List com Comments

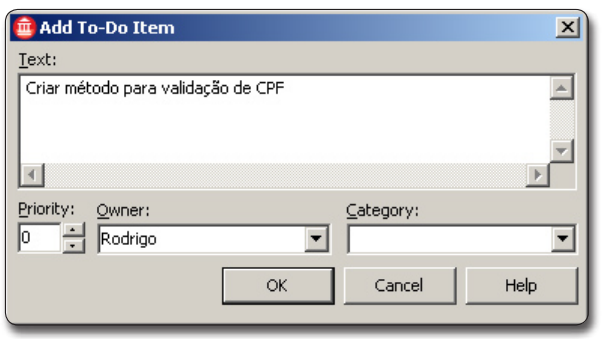


Figura 19. Uso de To-do item no Delphi XE

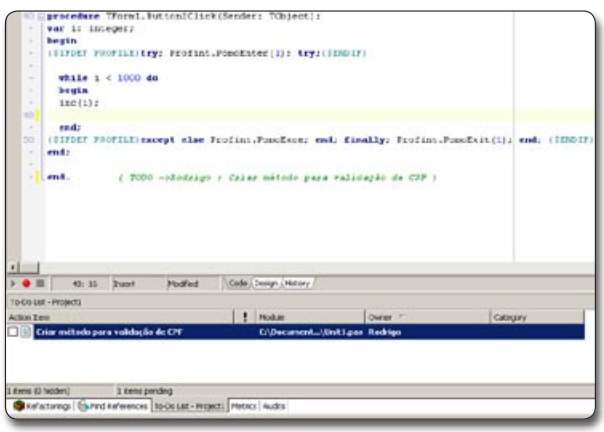


Figura 20. Janela de To-do List

deverá ser criado, modificado ou concluído a realização de tarefas. Preferencialmente o uso da palavra TODO indica algo novo a ser criado, HACK possibilita a alteração de uma tarefa e UNDONE informa que algo ainda não foi concluído. Além disso, a lista de tarefas pode ser ordenada pelas colunas do grid em sua janela.

Nota

A janela Quick Access é disponibilizada através de um complemento, extensão do Visual Studio 2010, chamado de Productivity Power Tools e está disponível gratuitamente para download no site do Visual Studio Gallery (vide seção Links no final do artigo).

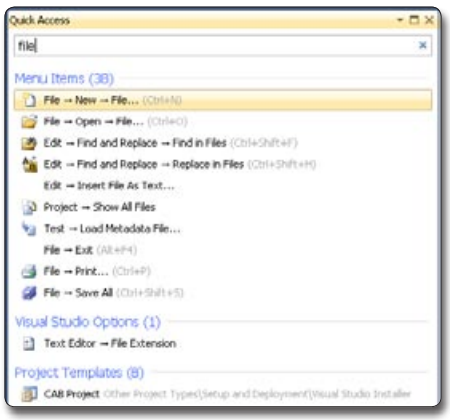


Figura 21. Uso de Quick Access

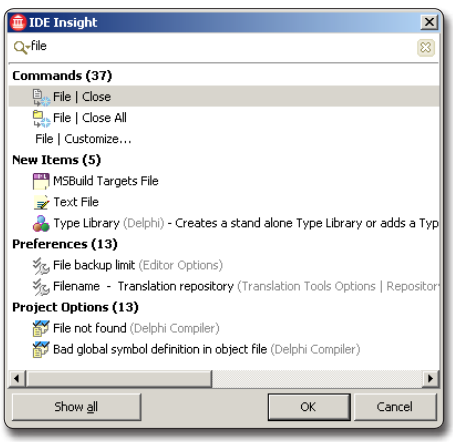


Figura 22. Uso de IDE Insight no Delphi XE

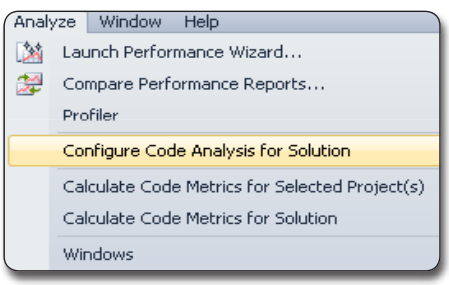


Figura 23. Menu de acesso ao Code Analysis

No Delphi XE as tarefas também estão presentes na forma de To-Do, possibilitando adicionar itens como visualizado na **Figura 19**, permitindo incluir uma descrição do que será adicionado, quem é seu criador, um nível de prioridade e uma categoria para a tarefa.

Além de adicionar itens de tarefas a serem realizadas, é possível verificar na janela de To-Do List o que está presente para posterior criação ou modificação. A **Figura 20** indica a janela com os itens adicionados, possibilidade de ordenação por determinada coluna do grid, filtros e abrir diretamente no local do código a ser modificado.

Quick Access

É uma nova janela de ferramentas disponível através da instalação de uma extensão para o IDE do Visual Studio 2010 que permite aos usuários realizarem pesquisas e executarem tarefas comuns sobre, principalmente, comandos de menus e executar operações de forma ágil através do teclado.

Com o Quick Access é possível executar comandos do main menu e do context menu, abrir a caixa de diálogo de um novo projeto, mostrar a janela de ferramentas, opções de navegação, navegar entre documentos abertos, criar e executar uma série de tarefas. Sua localização está presente no menu View através da opção Quick Access ou pelas teclas de atalho Control+3 e possibilitando realizar a pesquisa pelos termos que forem necessários e seus resultados serão apresentados logo abaixo e por categorias.

A **Figura 21** demonstra o uso de Quick Access. Ao pesquisar o termo "file", observe que o retorno das informações indica todas as opções de menu e templates que possuem a palavra file no seu conteúdo.

De maneira semelhante, no Delphi XE existe o IDE Insight que permite realizar tarefas semelhantes às descritas anteriormente pelo Quick Access. A **Figura 22** demonstra o seu uso utilizando a mesma palavra de pesquisa. A janela de IDE Insight do Delphi XE, acessada através da tecla de atalho F6, permite realizar operações relacionadas a qualquer opção de IDE sem a necessidade de realizar vários cliques do mouse, assim, acelerando a produtividade de código e de desenvolvimento.

Code Analysis

Realiza a análise detalhada para todos os módulos de código verificando determinadas regras de programação e de design definidas pelo Microsoft .NET Framework. É uma ferramenta que permite realizar uma verificação no código fonte, observando regras, uso de padrões, desempenho, segurança, melhores práticas para uso de namespaces, eventos, métodos e parâmetros, além de detectar problemas relativos aos itens anteriores.

O acesso para sua utilização com o Delphi Prism é realizado através do menu Analyze, demonstrado pela **Figura 23**, onde algumas opções de configuração e mapeamento podem ser realizadas. É possível realizar configurações através de um wizard indicando qual o aplicativo que será mapeado, iniciar um profiler para armazenamento de informações relativas à análise, configurações do próprio Code Analysis e cálculos de métricas para o projeto. Por final, após executar uma análise de performance a **Figura 24** demonstra os resultados através de gráfico e índices de desempenho.

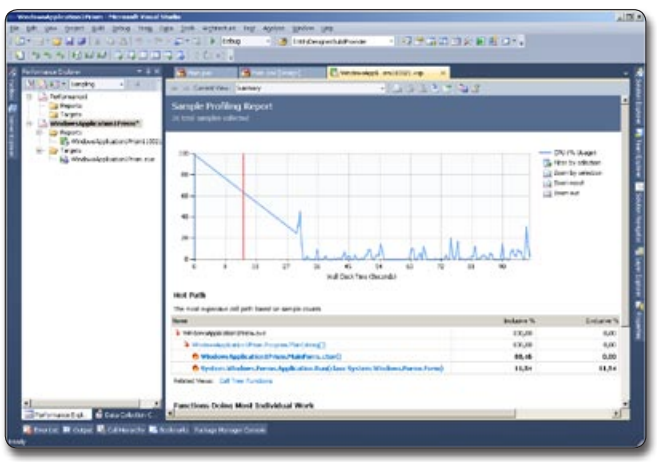


Figura 24. Demonstrativo de resultados

Pela imagem é observada na aba da esquerda a parte chamada de Performance Explorer que permite configurar e iniciar sessões de testes de desempenho para um aplicativo. Na outra

aba é apresentado o gráfico e seus índices mensurados pela execução do aplicativo através do profiler, as opções presentes neste profiler são indicadas na **Figura 25**. Através do menu anterior, poderá ser realizada uma nova sessão de monitoramento, adicionar ou remover itens, filtrar informações, importar e exportar filtros e salvar os dados analisados.

Ao utilizar o Delphi XE, uma opção semelhante está disponível através de QA Metrics, QA Audits ou ainda pelo uso de AQTime, possibilitando mensurar e analisar determinadas métricas de código e execução do aplicativo. A **Figura 26** demonstra o menu do AQTime com suas opções de uso. Opções semelhantes às utilizadas no Delphi Prism estão aqui disponíveis como o monitoramento, início de execução, término e alocação de recursos para Profiler. O AQtime é utilizado para múltiplas tarefas de otimização para melhorar o desempenho da aplicação e uso de memória.

Com os mesmos objetivos, QA Metrics busca auxiliar numa melhor forma de funcionamento e manutenção de um aplicativo visando obter um desenvolvimento de qualidade através de indicadores que irão demonstrar determinados pontos a serem considerados para o próprio desenvolvimento e tomada de decisões.

Além disso, QA Audits permite realizar monitoramento de código visando verificar determinados padrões de desenvolvimento seguindo modelos de especificações predeterminados, com isso, visando obter sistemas mais uniformes e com qualidade de desenvolvimento estruturada proporcionando melhor produtividade para a equipe de desenvolvimento.

Data

O menu data é o local responsável por apresentar alguns itens interessantes relacionados a dados, conexões com fontes de dados e ferramentas de manipulação de dados. A **Figura 27** indica as suas opções básicas. São acessadas algumas opções como a visualização de Data Sources que irão exibir as fontes de dados presentes no projeto. Além disso, permite criar relacionamentos entre fontes de dados e interfaces para o usuário, realizando a ligação entre esses controles.

A **Figura 28** demonstra o uso de um Data Source através de uma interface para o usuário representada por um Windows Forms. É observada a janela de Data Sources onde conexões a fonte de dados poderão ser realizadas, no exemplo, uma conexão ao banco de dados SQL Server foi realizada e vinculada à tabela Aluno, possuindo alguns atributos. Para o Windows Forms, simplesmente foi arrastado a tabela aluno da janela da esquerda para a direita, apresentando a visualização conforme a imagem demonstra, procedimento este, realizado de forma bastante semelhante no Delphi XE.

Outra opção de menu é a adição de um novo Data Source, uma fonte de dados que poderá ser vinculada a um banco de dados, um objeto, um serviço ou ainda um Sharepoint que retorne algum tipo de dado para a aplicação.

Além de Data Sources, apresenta comparação de esquemas, também chamado de Schema Compare, realizando um comparativo estrutural entre duas fontes de dados distintas e a comparação de dados, através de Data Compare.

Outra funcionalidade de bastante importância é o Transact-SQL Editor possibilitando realizar consulta a fontes de dados distintas através da linguagem de consultas SQL. A **Figura 29** mostra o uso de uma consulta SQL acessando o banco de dados Northwind. É visualizado o Transact-SQL Editor e a construção de uma consulta SQL que irá retornar os dados da tabela Employees, observe que o banco de dados acessado é o Northwind e os resultados da consulta estão dispostos no grid.

O Transact-SQL pode ser utilizado para criar e executar consultas, modificar objetos de banco de dados e scripts, realizar atualizações no banco e esquemas de dados e ainda pode ser acessado de diversos locais do próprio ambiente de desenvolvimento. Além disso, ao iniciar a janela do Transact-SQL um menu de opções representado pela **Figura 30** é apresentado. No menu é possível realizar novas conexões a fontes de dados, executar uma consulta SQL, validar, demonstrar a estimativa de execução da consulta, habilitar ou desabilitar

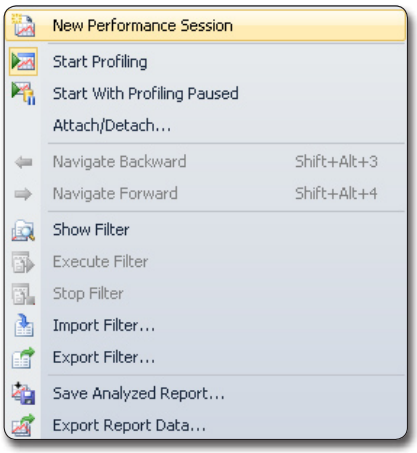


Figura 25. Menu do Profiler

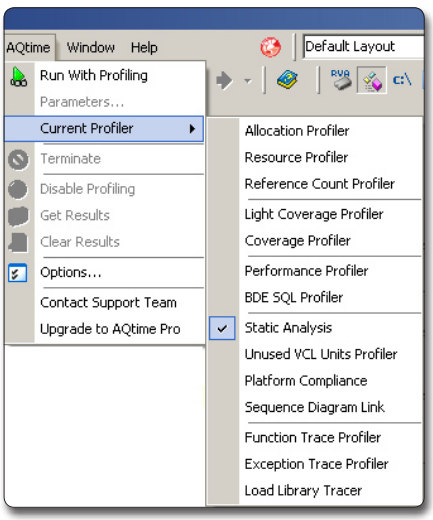


Figura 26. Menu do Profiler no Delphi XE

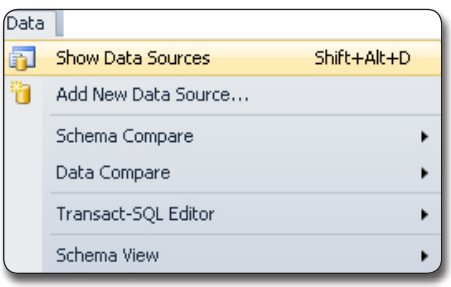


Figura 27. Menu Data

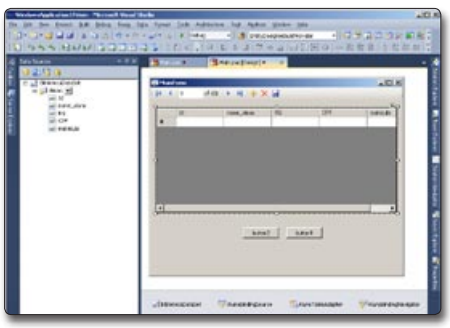


Figura 28. Data Sources e um formulário Windows

o IntelliSense ao informar a consulta, mostrar ou ocultar os resultados da consulta no grid.

Ao utilizar o Delphi XE, o acesso e visualização a dados é realizado através do Data Explorer, da mesma forma, foi criada uma conexão ao banco de dados SQL Server, acessada a instância Northwind e a tabela Employees, conforme a **Figura 31** traz o acesso aos dados e a sua visualização através do grid. Além de visualizar os dados é permitida a alteração das informações como LastName e FirstName, alteração da estrutura da tabela pelo Data Explorer, adição e exclusão de atributos e a exclusão de uma tabela (drop).

Nota

Microsoft SharePoint é uma plataforma de colaboração voltada para aplicações através de um site que poderá conter recursos colaborativos semelhantes a fóruns, documentos compartilhados, versionamento de arquivos entre outros.

Conclusão

O Delphi Prism faz uso do ambiente presente no Visual Studio 2010 e de suas funcionalidades, possibilitando criar aplicações robustas e com tecnologia recente. As funcionalidades disponíveis apresentam uma expressão visual destacada, auxiliando e facilitando o desenvolvimento.

Durante o decorrer desta série de artigos, vários itens, janelas, opções e ferramentas foram relacionadas, sempre tentando esclarecer e diminuir o impacto visual de quem já utilizava o Delphi e, no momento, gostaria de iniciar a produzir com o Delphi Prism e seu ambiente diferenciado em alguns aspectos.

Neste artigo, foram demonstrados menus e suas opções, principalmente, voltadas ao projeto, aos melhoramentos na fase de confecção de código fonte e a estruturação e análise de conexão a fonte de dados distintas, sempre visando realizar comparativos com o Delphi XE, para uma melhor transição de ambientes de programação.

O principal objetivo é auxiliar a produção de sistemas e melhorar a qualidade do que será realizado. Todas as funcionalidades descritas seguiram um padrão de comparação e relacionamento entre os dois ambientes, tentando adaptar de forma simples e clara ao novo ambiente disponibilizado pelo Delphi Prism.

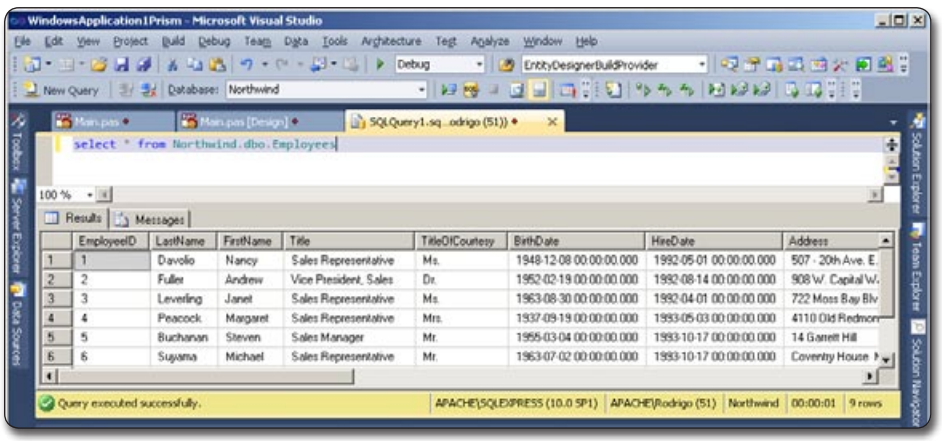


Figura 29. Transact-SQL Editor

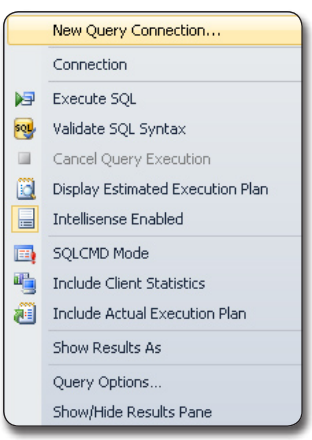


Figura 30. Menu para o Transact-SQL Editor

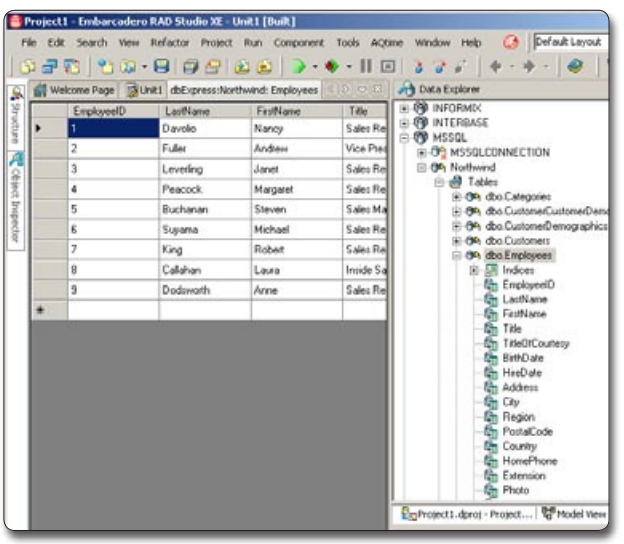


Figura 31. Acesso à fonte de dados e visualização de informações

Rodrigo Araujo
rodrigo.arsi@gmail.com
Bacharel em Ciência da Computação pela UFSM – Santa Maria/RS trabalha com Análise e Desenvolvimento de Software há mais de 10 anos. Desde 2003, utiliza tecnologias .NET.

Delphi Prism Wiki
http://prismwiki.codegear.com

Win32 Delphi vs. Delphi Prism
http://prismwiki.codegear.com/en/Win32_Delphi_vs._Delphi_Prism

Delphi Prism XE
http://www.embarcadero.com/products/delphi-prism

Delphi XE
http://www.embarcadero.com/products/delphi

VCL and RTL Changes for XE - RAD Studio XE
http://docwiki.embarcadero.com/RADStudio/XE/en/VCL_and_RTL_Changes_for_XE

Visual Studio Gallery
http://visualstudiogallery.msdn.microsoft.com

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Hospede seus sites de forma simples, rápida e ilimitada.

Plano **Básico**

- 1 domínio principal
- 50 domínios hospedados
- Transferência **ilimitada**
- Espaço **ilimitado**
- 10 contas de e-mail
- PHP, ASP e ASP.NET
- Painel de controle Plesk
- Suporte 24hs

Tudo isso por apenas

R$ **15,90** por mês
pagamento trimestral

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

- Conheça também nossos planos profissionais com registro de domínio grátis e isenção de taxas anuais enquanto for cliente.

HOSPEDIX
HOSPEDAGEM PROFISSIONAL
www.hospedix.com.br

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

RVSTMEDIA

As promoções anunciadas não são cumulativas com outras promoções e são válidas apenas para clientes Hospedix ativos.

Microsoft

wunderman

Quem desenvolve em múltiplas linguagens pensa em Windows Azure.

Windows Azure é uma plataforma de desenvolvimento que permite desenvolver e hospedar aplicativos na Nuvem. Você pode programar em múltiplas linguagens e tecnologias, como .NET, PHP, JAVA, RUBY e PHYTON, além de ter a possibilidade de atualizar seus aplicativos de uma forma muito mais rápida e prática, sem sair do ar. Possibilidade de trabalhar de qualquer lugar, em várias linguagens. Este é o Poder da Nuvem.

Teste o Windows Azure. É grátis.
Acesse Microsoft.com.br/nuvem/desenvolvedor

Para ler o código ao lado, baixe grátis nosso aplicativo para celular em http://gettag.mobi e acesse o site mobile. Para assinar esta oferta será necessário um cartão de crédito (nacional ou internacional). Ao exceder os montantes definidos* haverá cobrança dos serviços com base na taxa regular. Esta oferta está disponível até o dia 30 de setembro de 2011 (23h59 UTC).

*Mais informações: http://www.microsoft.com.br/nuvem/desenvolvedor

Windows Azure

Microsoft é marca ou marca registrada da Microsoft Corporation nos Estados Unidos e/ou em outros países.